Probabilistic Programming and Artificial Intelligence

Vikash Mansinghka

MIT Probabilistic Computing Project

http://probcomp.csail.mit.edu

Acknowledgements

The Probabilistic Computing Project:

Standing Row (L to R): **Feras Saad**, **Marco Cusumano-Towner**, Jonathan Rees, Sara Rendtorff-Smith, Josh Thayer, Zane Shelby, **Ulrich Schaechtle**
Seated Row (L to R): Vikash Mansinghka, Amanda Brower, Desiree Dudley, Cameron Freer, Alex Lew, Tim Trautman

With the fiscal support of:

DARPA

Office of Naval Research Science & Technology

Army Research Office

intel

Google

Outline

1. **Motivation**
2. **What is probabilistic programming?**

   Pedagogical example: simple (or not-so-simple) curve fitting

3. **Programmable inference, not just black-box**

   Application: machine perception via inverse graphics

4. **Learning the structure and parameters of probabilistic programs**

   Application: automatic data modeling for scientific data analysis

5. **The MIT Modeling and Inference Stack**

Exuberance about machine learning and "big data"

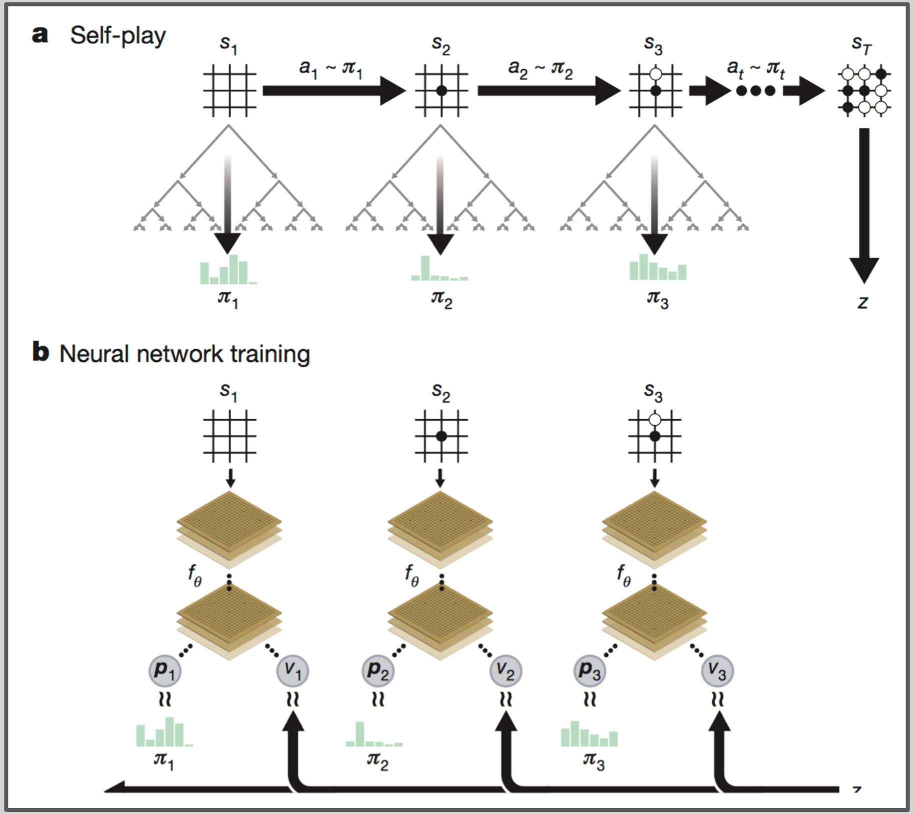
WIRED MAGAZINE: 16.07

SCIENCE : DISCOVERIES

The End of Theory: The Data Deluge Makes the Scientific Method Obsolete

By Chris Anderson 06.23.08

Machine learning success story: AlphaGo Zero



Silva et al., Nature 2017

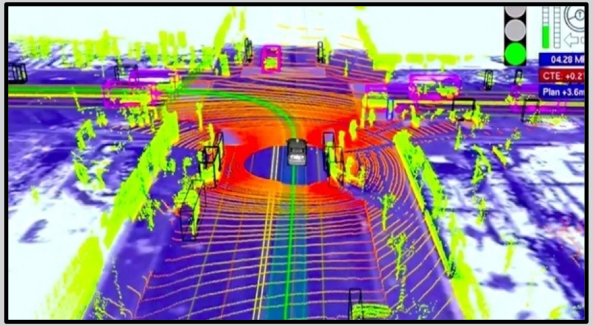
The limitations of machine learning

Go

same rules for ~2,500 years

one winner, one loser

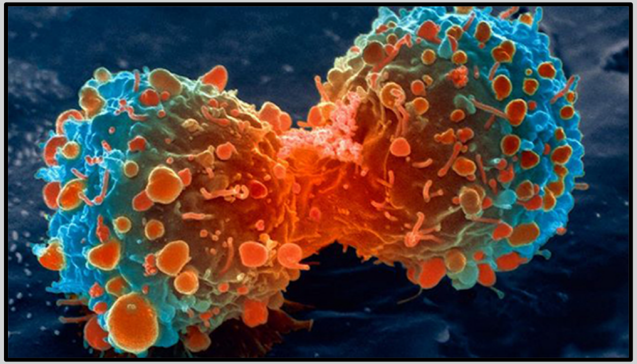
Autonomous driving



simulations are available, but environment varies widely

drivers and pedestrians have complex & conflicting objectives

Cancer

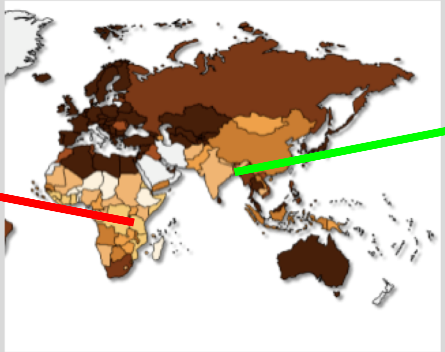


every cancer cell is different

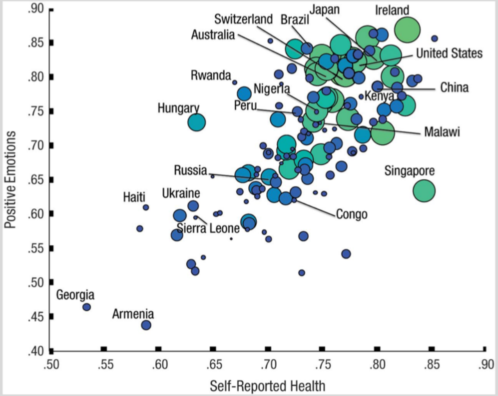
treatment requires life-and-death tradeoffs

Challenge #1: Machine common-sense, at the level of an 18-month-old

Challenge #2: Machine expert systems that help human experts collaboratively interpret empirical data



Data



Prior knowledge from:

- Epidemiologists
- Economists
- Field workers
- Policy advocates
- Stakeholders

What we need

Intelligence is not just about *pattern recognition*.

It is about *modeling the world*…

- *explaining* and *understanding* what we see.
- *imagining* things we could see but haven't yet.
- *making judgment calls* in ambiguous situations.
- *problem solving* and *planning* actions to make these things real.
- *building new models* as we learn more about the world.
- *sharing our models* with each other, via language.

Outline

1. **Motivation**

2. **What is probabilistic programming?**

    Pedagogical example: simple (or not-so-simple) curve fitting

3. **Programmable inference, not just black-box**

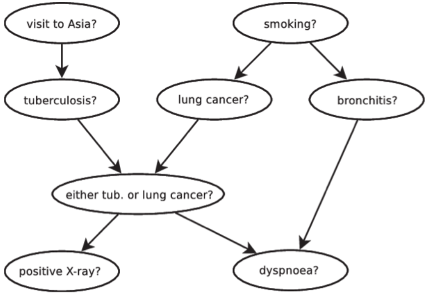
    Application: machine perception via inverse graphics

4. **Learning the structure and parameters of probabilistic programs**

    Application: automatic data modeling for scientific data analysis

5. **The MIT Modeling and Inference Stack**

The need for probabilistic programming



Causal models

```
(DEFUN SUMLISTS (A B SUMMED)
  (COND
   ((OR (NULL A) (NULL B))
     SUMMED)
   ('T
    (SUMLISTS (CDR A) (CDR B)
    (APPEND SUMMED
    (LIST (+ (CAR A)(CAR B))))))))
```

Symbolic programs

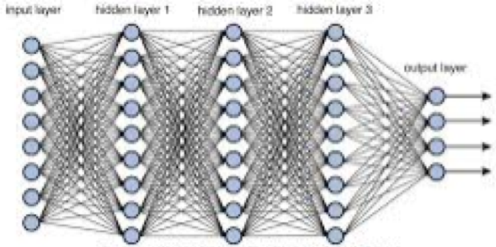
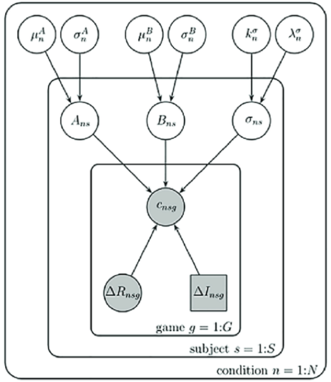


Figure 12.2 Deep network architecture with multiple layers.

Deep neural networks



Priors

$\mu_n^A \sim \text{Gaussian}(0, 1000)$

$\sigma_n^A \sim \text{Gamma}(0.001, 0.001)$

$\mu_n^B \sim \text{Gaussian}(0, 1000)$

$\sigma_n^B \sim \text{Gamma}(0.001, 0.001)$

$k_n^\sigma \sim \text{Exponential}(0.001)$

$\lambda_n^\sigma \sim \text{Exponential}(0.001)$

Subject specific parameters

$A_{ns} \sim \text{Gaussian}(\mu_n^A, \sigma_n^A)$

$B_{ns} \sim \text{Gaussian}(\mu_n^B, \sigma_n^B)$

$\sigma_{ns} \sim \text{Gamma}(k_n^\sigma, \lambda_n^\sigma)$

Observed choices

$p_{nsg} \leftarrow \left[1 + \exp\left(\frac{\Delta R_{nsg} + A_{ns}\Delta I_{nsg} + B_{ns}}{\sigma_{ns}}\right)\right]^{-1}$

$c_{nsg} \sim \text{Bernoulli}\,(p_{nsg})$

Hierarchical Bayesian models

What is probabilistic programming?

Two technical ideas:

1. Models can be represented using programs that make stochastic choices

2. Operations on models can be represented as meta-programs

What is probabilistic programming?

Two technical ideas:

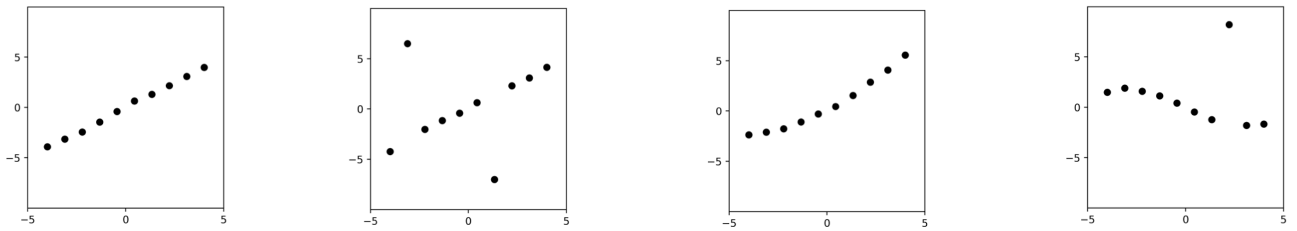
1. **Models can be represented using programs that make stochastic choices**

2. **Operations on models can be represented as meta-programs**

    Inference - finding probable values for latent variables
    Learning - finding probable model parameters and structure models given data
    Querying - making predictions for previously unseen data, given a model
    Analysis - estimating the amount of information between variables in a model

Curve fitting with model selection and outlier detection

Four data sets

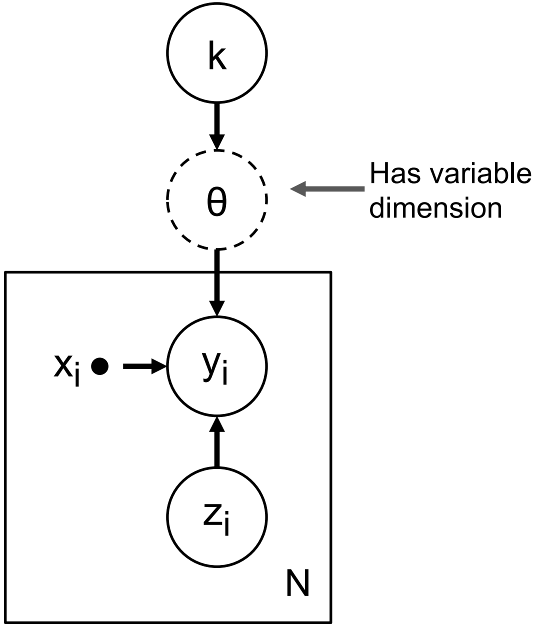


$$k \sim \text{Uniform}(\{1,2,3,4\})$$ // Choose degree of polynomial

$$\boldsymbol{\theta} \sim \text{Normal}(\mathbf{0}_{k+1}, \mathbf{I}_{k+1})$$ // Choose coefficients

$$z_i \sim \text{Bernoulli}(0.1) \text{ for } i = 1 \ldots N$$ // Choose outlier assignments

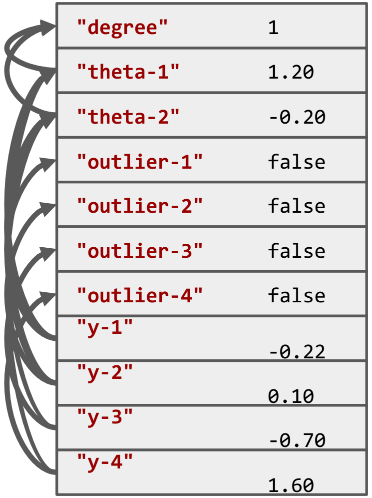
$$y_i \sim \begin{cases} \text{Normal}(\sum_{j=1}^{k+1} x_i^{j-1}\theta_j, 1) & \text{if } z_i = 0 \\ \text{Normal}(\sum_{j=1}^{k+1} x_i^{j-1}\theta_j, 10) & \text{if } z_i = 1 \end{cases} \quad \text{for } i = 1 \ldots N$$



As a graphical model

```
@probabilistic function model(x::Vector{Float64})

    # prior over degree of polynomial
    degree_prior = [0.25, 0.25, 0.25, 0.25]

    # generate degree (either 1, 2, 3, or 4)
    degree = @choice(categorical(degree_prior), "degree")

    # generate parameters
    parameters = Vector{Float64}(degree+1)
    for k=1:(degree+1)
        parameters[k] = @choice(normal(prior_mean, prior_std), "theta-$k")
    end

    # generate data
    y = Vector{Float64}(length(x))
    for i=1:length(x)
        if degree == 1
            y_mean = dot(parameters, [1., x[i]])
        elseif degree == 2
            y_mean = dot(parameters, [1., x[i], x[i]^2])
        elseif degree == 3
            y_mean = dot(parameters, [1., x[i], x[i]^2, x[i]^3])
        else
            y_mean = dot(parameters, [1., x[i], x[i]^2, x[i]^3, x[i]^4])
        end
        is_outlier = @choice(flip(prob_outlier), "outlier-$i")
        noise = is_outlier ? outlier_noise : inlier_noise
        y[i] = @choice(normal(y_mean, noise), "y-$i")
    end
end
```

As a probabilistic program



One possible **execution trace** of the program

with input `x = [-3, 0, 2, 3]`
and output `y = [-0.22, 0.1, -0.70, 1.60]`

Inference in a probabilistic program

```
(trace, weight) = query(program, args, observations)
```

$\mathbf{x}$ $\quad\quad$ ξ $\quad\quad$ $\mathcal{P}$ $\quad\quad$ α $\quad\quad$ $\mathbf{y}$

Distribution on traces induced by executing program
(e.g. the prior)

$$p(\mathbf{x}; \mathcal{P}, \alpha)$$

Distribution on traces conditioned on observations
(e.g. the posterior)

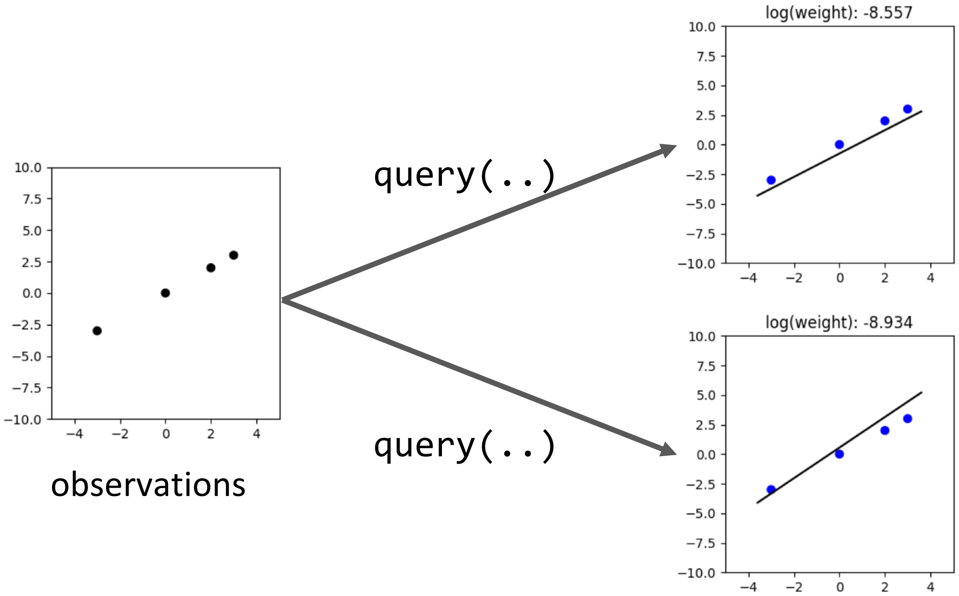
$$p(\mathbf{x}|\mathbf{y}; \mathcal{P}, \alpha) \propto p(\mathbf{x}; \mathcal{P}, \alpha) \prod_{i \in \mathbf{y}} \delta(x_i, y_i)$$

Distribution on traces sampled during query execution
(e.g. the posterior approximation)

$$q(\mathbf{x}; \mathcal{P}, \alpha, \mathbf{y}) \approx p(\mathbf{x}|\mathbf{y}; \mathcal{P}, \alpha)$$

Querying a probabilistic program

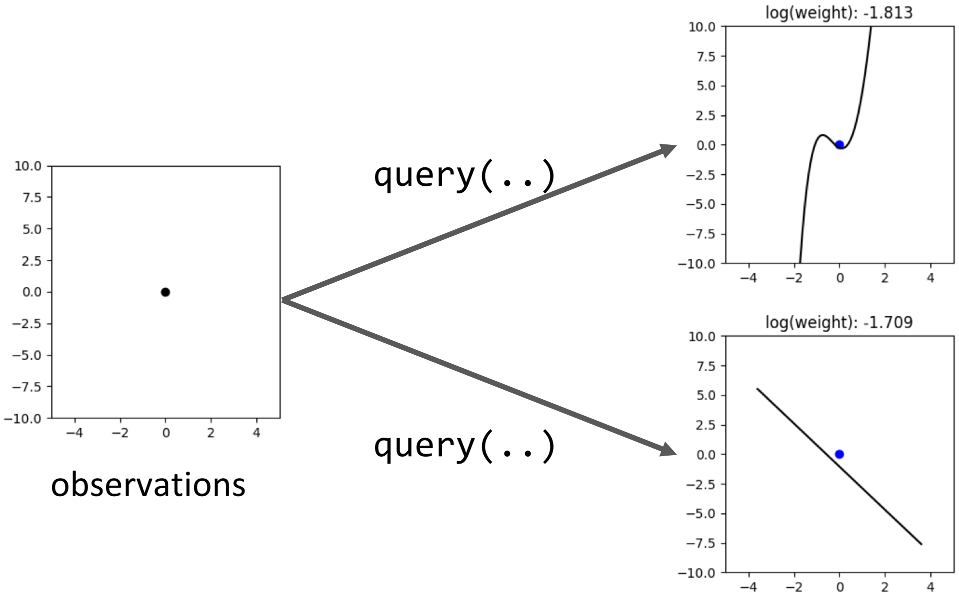
```
observations = Trace()
observations["y-1"] = -3.0
observations["y-2"] =  0.0
observations["y-3"] =  2.0
observations["y-4"] =  3.0
(trace, weight) = query(model, ([-3, 0, 2, 3],), observations)
```



Querying a probabilistic program

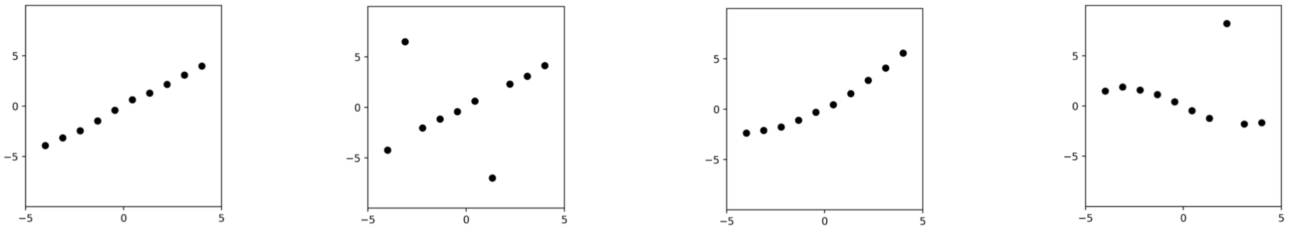
Observing a single data point

```
observations = Trace()
observations["y-2"] = 0.0
(trace, weight) = query(model, ([-3, 0, 2, 3],), observations)
```



Approximate posterior samples

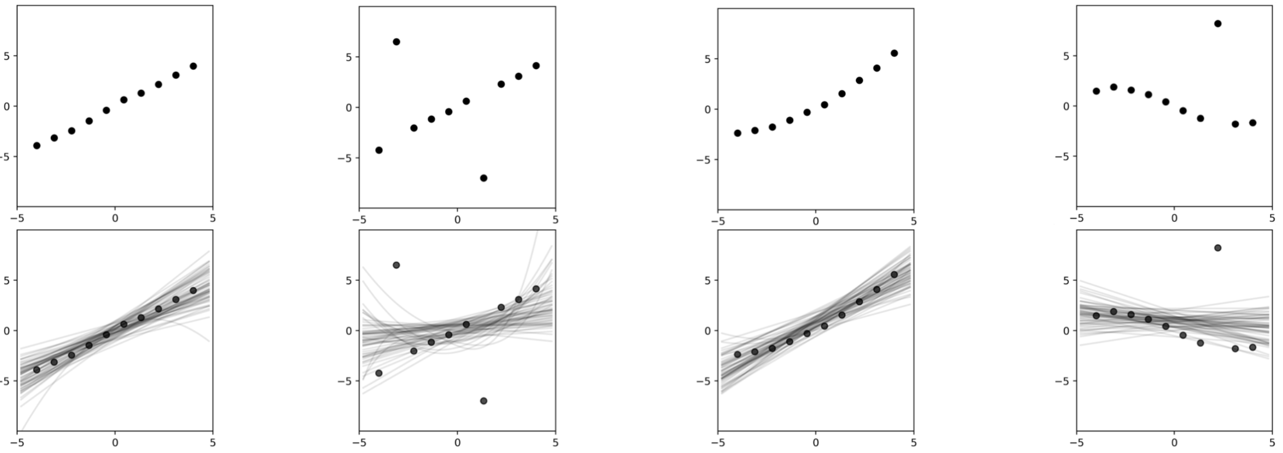
Four data sets



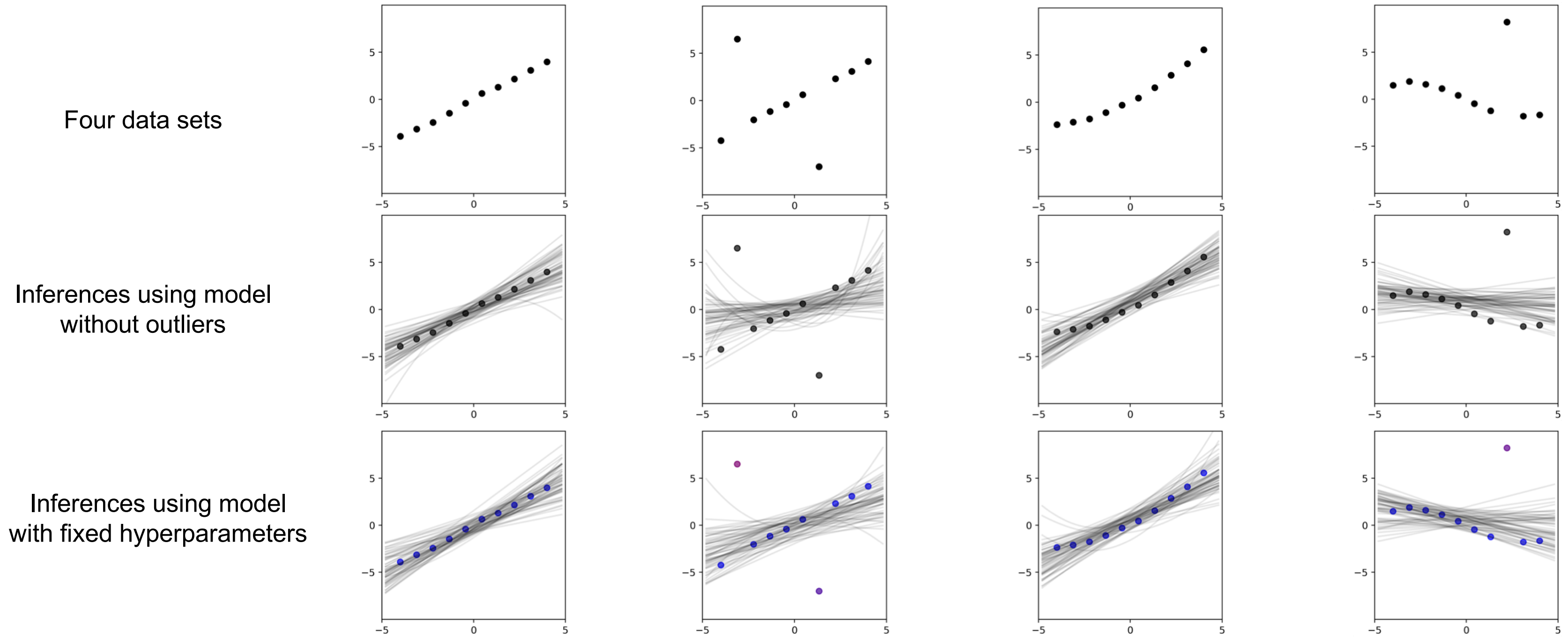
Approximate posterior samples

Four data sets

Inferences using model
without outliers

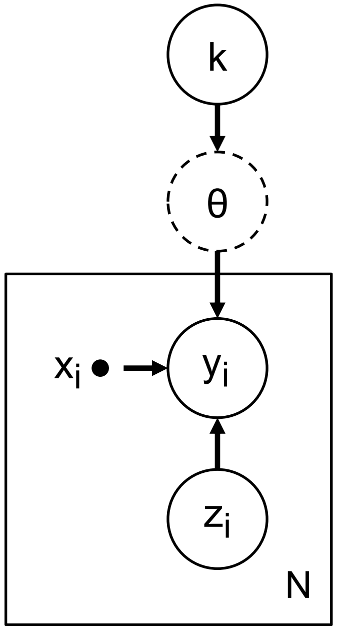


Approximate posterior samples



Adding hyperparameter uncertainty

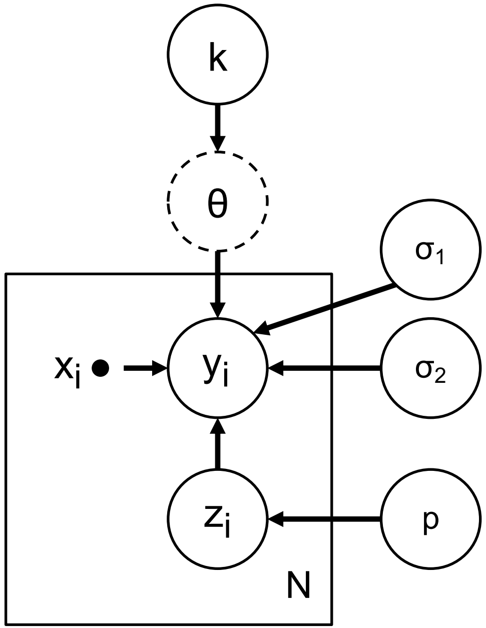
$$k \sim \mathrm{Uniform}(\{1,2,3,4\})$$
$$\boldsymbol{\theta} \sim \mathrm{Normal}(\mathbf{0}_{k+1}, \mathbf{I}_{k+1})$$
$$z_i \sim \mathrm{Bernoulli}(0.1) \text{ for } i = 1 \ldots N$$
$$y_i \sim \begin{cases} \mathrm{Normal}(\sum_{j=1}^{k+1} x_i^{j-1}\theta_j, 1) & \text{if } z_i = 0 \\ \mathrm{Normal}(\sum_{j=1}^{k+1} x_i^{j-1}\theta_j, 10) & \text{if } z_i = 1 \end{cases} \quad \text{for } i = 1 \ldots N$$



Model with fixed hyperparameters

Adding hyperparameter uncertainty

$$k \sim \text{Uniform}(\{1,2,3,4\})$$
$$\boldsymbol{\theta} \sim \text{Normal}(\mathbf{0}_{k+1}, \mathbf{I}_{k+1})$$
$$p \sim \text{Beta}(1,20)$$
$$\sigma_1 \sim \text{Gamma}(2,1)$$
$$\sigma_2 \sim \text{Gamma}(1,20)$$
$$z_i \sim \text{Bernoulli}(p) \text{ for } i = 1 \ldots N$$
$$y_i \sim \begin{cases} \text{Normal}(\sum_{j=1}^{k+1} x_i^{j-1}\theta_j, \sigma_1) & \text{if } z_i = 0 \\ \text{Normal}(\sum_{j=1}^{k+1} x_i^{j-1}\theta_j, \sigma_2) & \text{if } z_i = 1 \end{cases} \quad \text{for } i = 1 \ldots N$$



Model with hyperparameter uncertainty

Adding hyperparameter uncertainty

```
@probabilistic function model(x::Vector{Float64})

    # prior over degree of polynomial
    degree_prior = [0.25, 0.25, 0.25, 0.25]

    # generate degree (either 1, 2, 3, or 4)
    degree = @choice(categorical(degree_prior), "degree")

    # generate parameters
    parameters = Vector{Float64}(degree+1)
    for k=1:(degree+1)
        parameters[k] = @choice(normal(0, 1), "theta-$k")
    end

    # generate data
    y = Vector{Float64}(length(x))
    for i=1:length(x)
        if degree == 1
            y_mean = dot(parameters, [1., x[i]])
        elseif degree == 2
            y_mean = dot(parameters, [1., x[i], x[i]^2])
        elseif degree == 3
            y_mean = dot(parameters, [1., x[i], x[i]^2, x[i]^3])
        else
            y_mean = dot(parameters, [1., x[i], x[i]^2, x[i]^3, x[i]^4])
        end
        is_outlier = @choice(flip(0.1), "outlier-$i")
        noise = is_outlier ? 10.0 : 1.0
        y[i] = @choice(normal(y_mean, noise), "y-$i")
    end
end
```

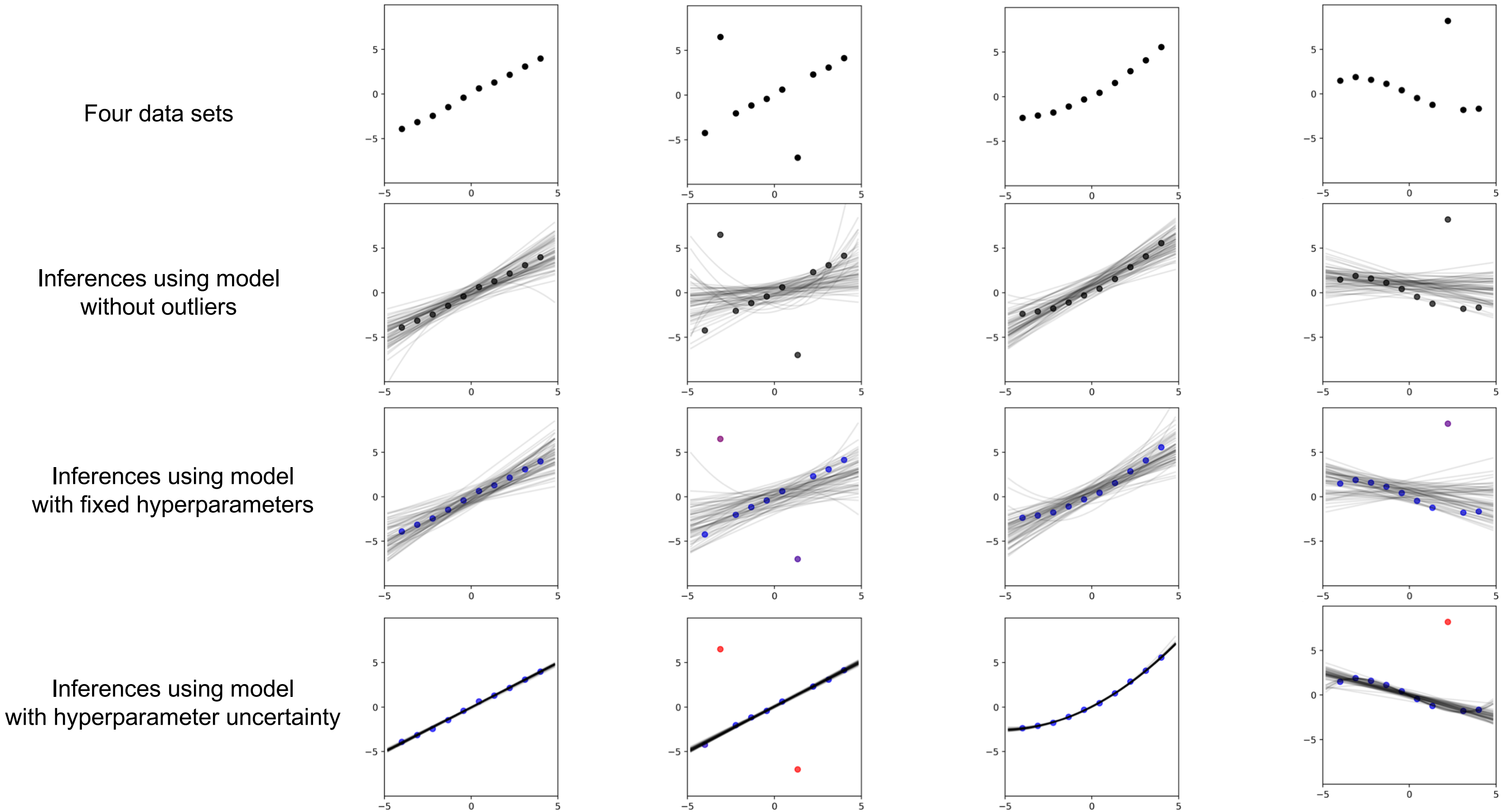
Model with fixed hyperparameters

Adding hyperparameter uncertainty

```
@probabilistic function model(x::Vector{Float64})

    # prior over degree of polynomial
    degree_prior = [0.25, 0.25, 0.25, 0.25]

    # generate degree (either 1, 2, 3, or 4)
    degree = @choice(categorical(degree_prior), "degree")

    # generate parameters
    parameters = Vector{Float64}(degree+1)
    for k=1:(degree+1)
        parameters[k] = @choice(normal(0, 1), "theta-$k")
    end

    # hyperparameters
    inlier_noise = @choice(gamma(2., 1.), "inlier-noise")
    outlier_noise = @choice(gamma(10., 1.), "outlier-noise")
    prob_outlier = @choice(beta(1., 20.), "prob-outlier")

    # generate data
    y = Vector{Float64}(length(x))
    for i=1:length(x)
        if degree == 1
            y_mean = dot(parameters, [1., x[i]])
        elseif degree == 2
            y_mean = dot(parameters, [1., x[i], x[i]^2])
        elseif degree == 3
            y_mean = dot(parameters, [1., x[i], x[i]^2, x[i]^3])
        else
            y_mean = dot(parameters, [1., x[i], x[i]^2, x[i]^3, x[i]^4])
        end
        is_outlier = @choice(flip(prob_outlier), "outlier-$i")
        noise = is_outlier ? outlier_noise : inlier_noise
        y[i] = @choice(normal(y_mean, noise), "y-$i")
    end
end
```

Model with hyperparameter uncertainty

Approximate posterior samples



Outline

1. **Motivation**
2. **What is probabilistic programming?**

   Pedagogical example: simple (or not-so-simple) curve fitting

3. **Programmable inference, not just black-box**

   Application: machine perception via inverse graphics

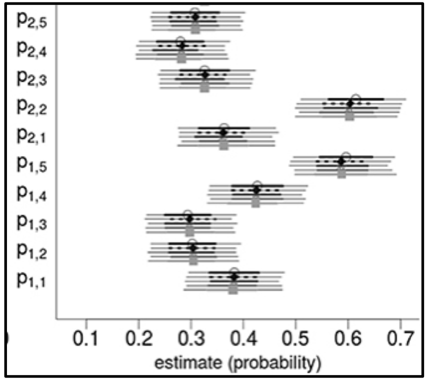
4. **Learning the structure and parameters of probabilistic programs**

   Application: automatic data modeling for scientific data analysis

5. **The MIT Modeling and Inference Stack**

Probabilistic models and inference algorithms

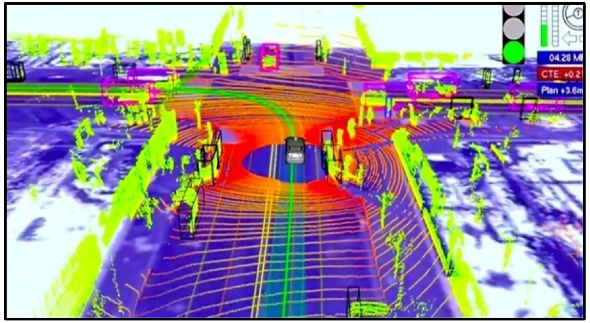
Statistics



Model: numerical effect sizes

Algorithm: Markov chain Monte Carlo inference to quantify uncertainty

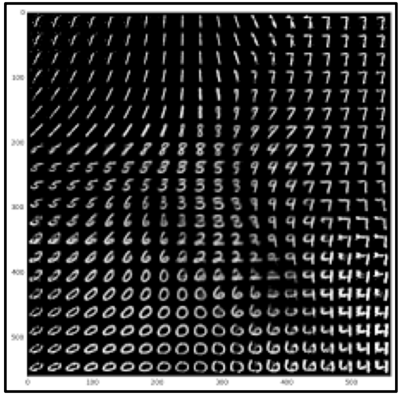
Robotics



Model: tracks of vehicle & people

Algorithm: Particle filter to track small changes over time

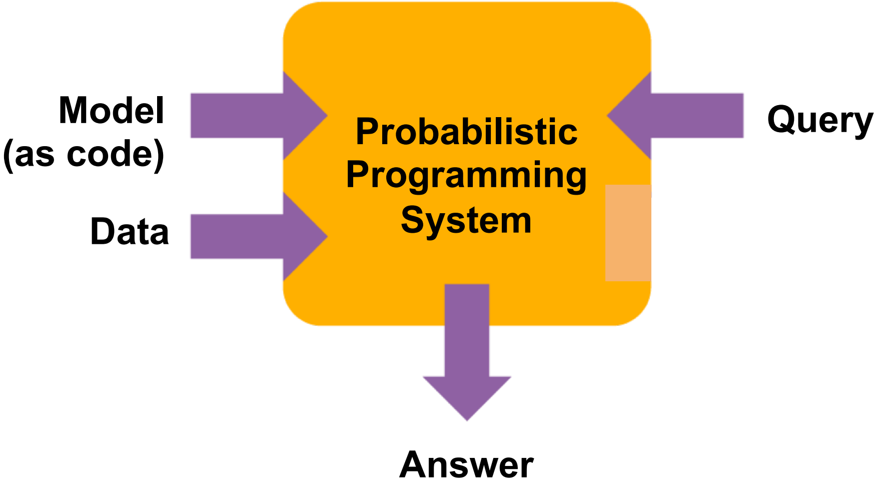
Machine learning



Model: neural network parameters

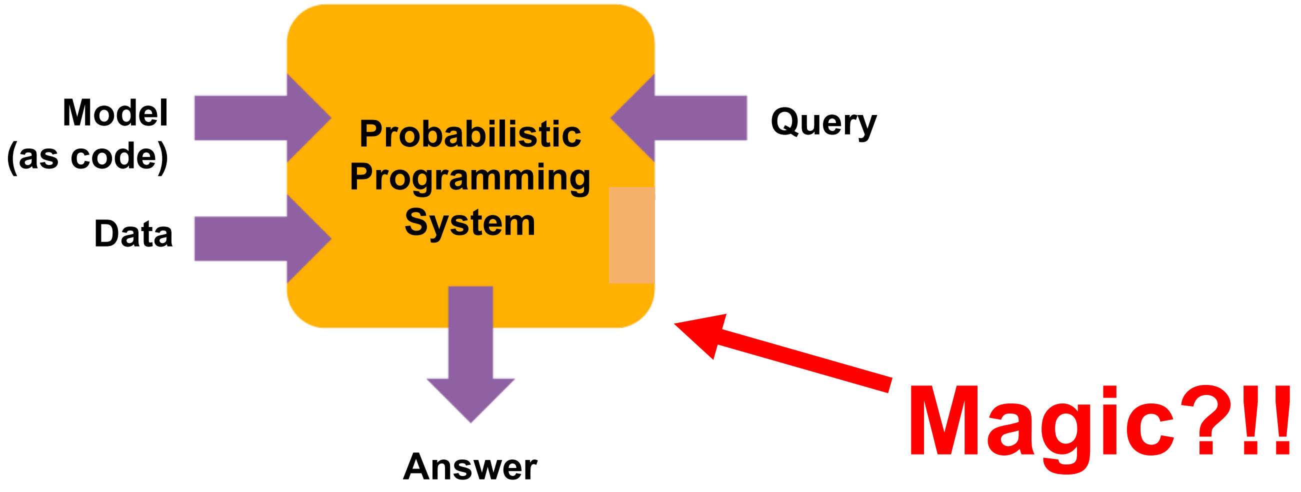
Algorithm: ``best'' parameters found by stochastic gradient descent

Probabilistic programming



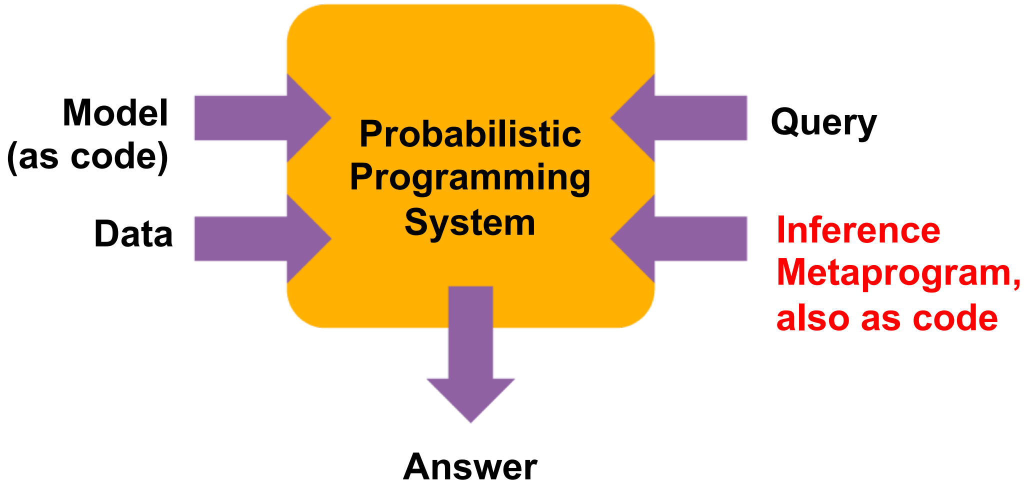
See e.g. Church, published in Goodman*, Mansinghka*, et al. (2008)

Probabilistic programming with programmable inference



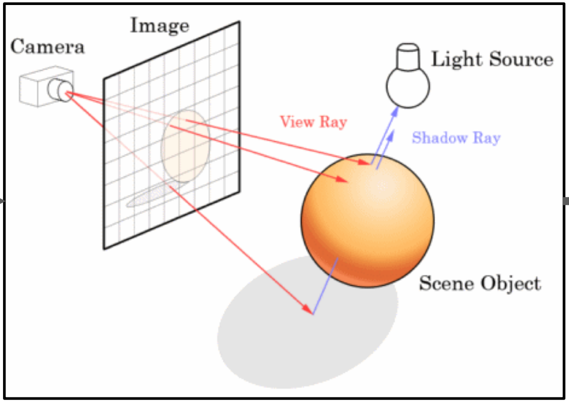
See e.g. Church (Goodman*, Mansinghka*, et al. [2008]), Prolog, ...

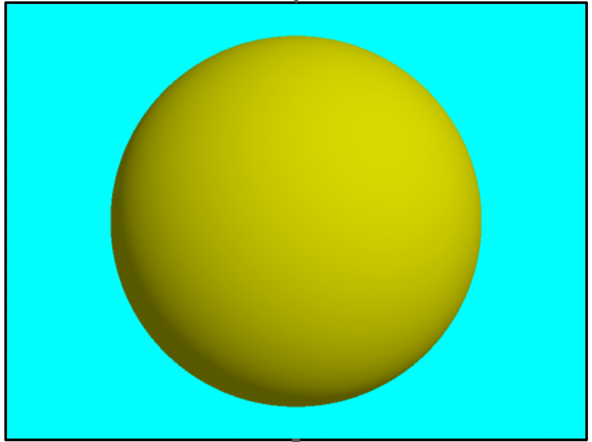
Probabilistic programming with programmable inference



Application: machine perception as inverse graphics

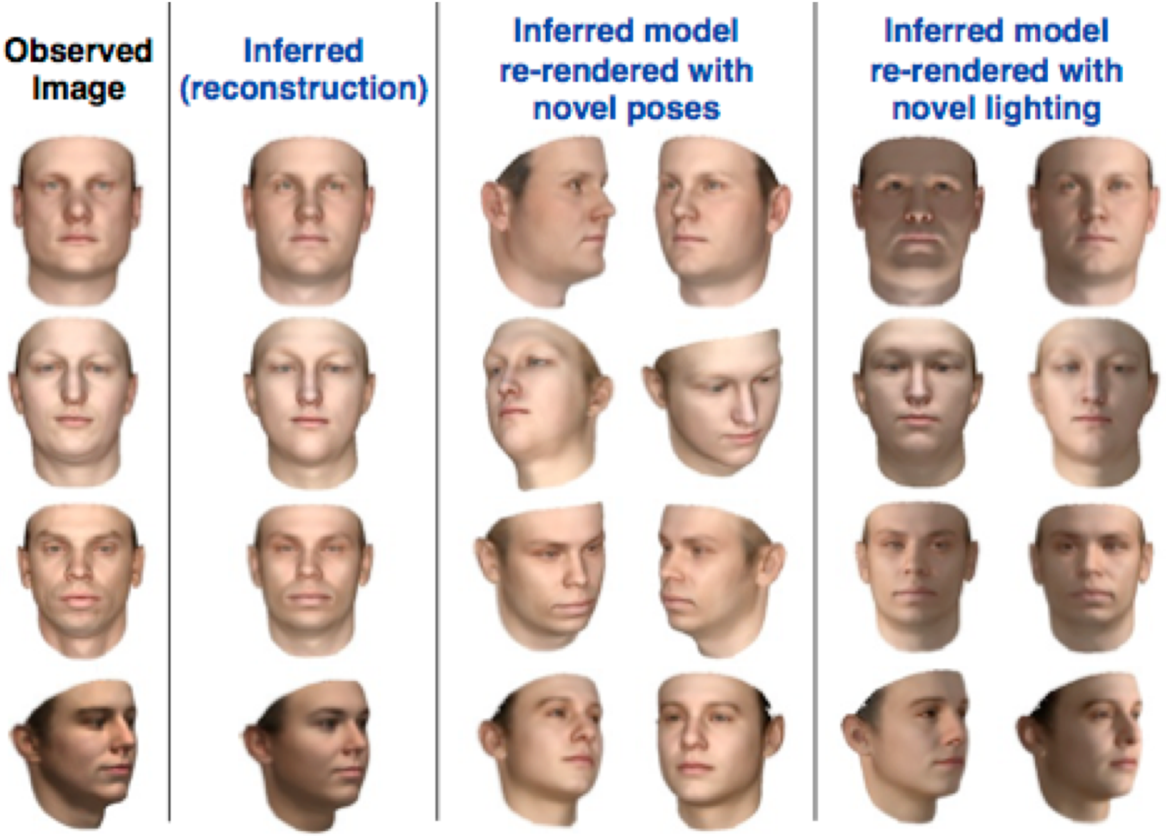
```
#include "colors.inc"
 background { color Cyan }
 camera {
   location <0, 2, -3>
   look_at  <0, 1,  2>
 }
 sphere {
   <0, 1, 2>, 2
   texture {
     pigment { color Yellow }
   }
 }
  light_source  {  <2,  4,  -3>
color White}
```



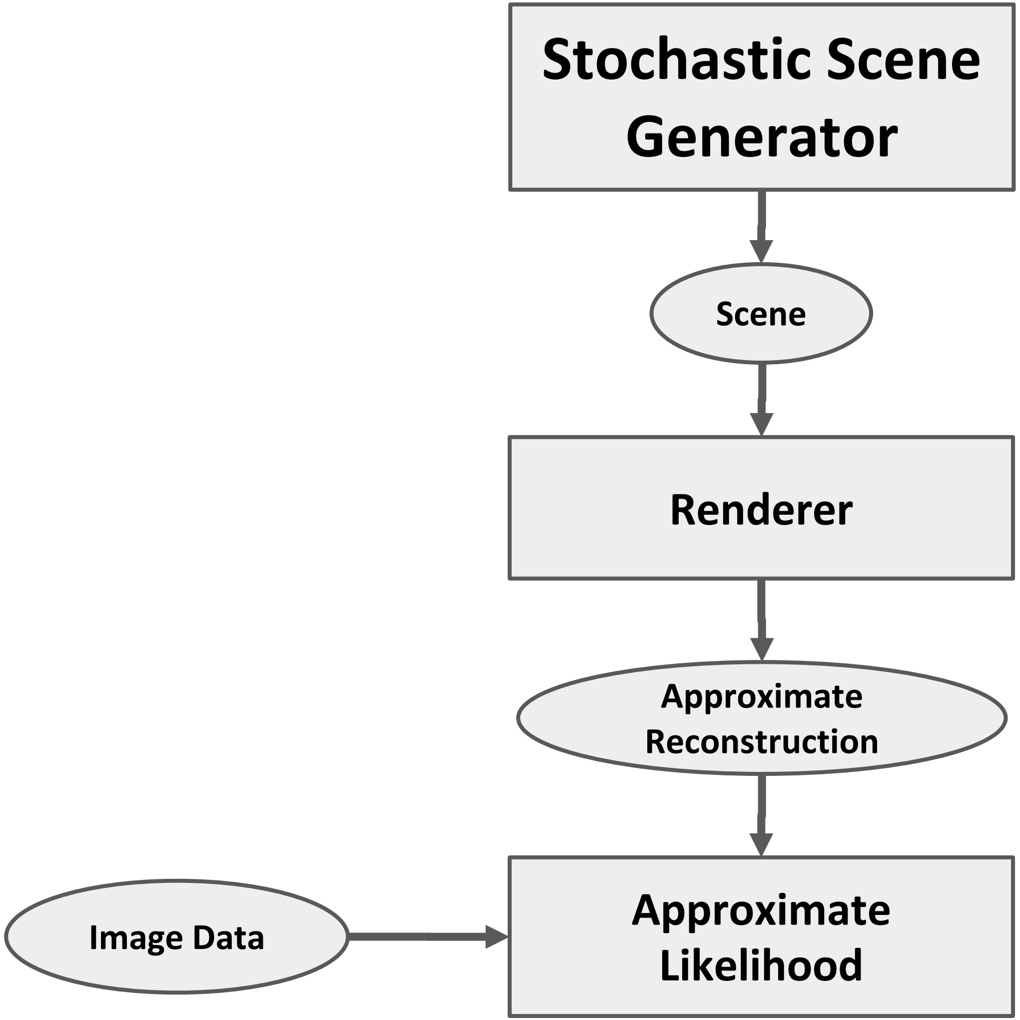


?

"What does this face look like from the side? Or when lit differently?"

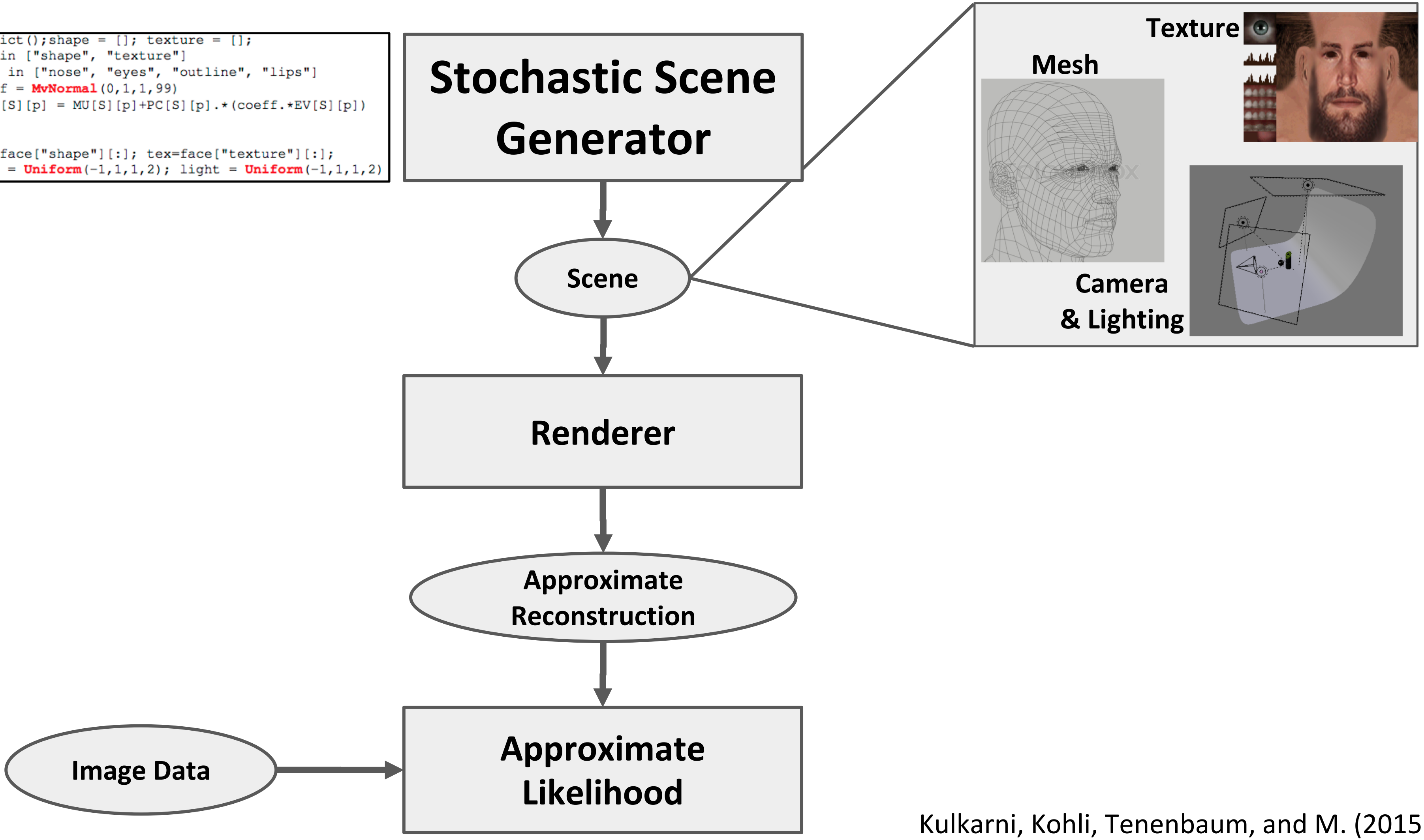


Kulkarni, Kohli, Tenenbaum, and Mansinghka (2015)



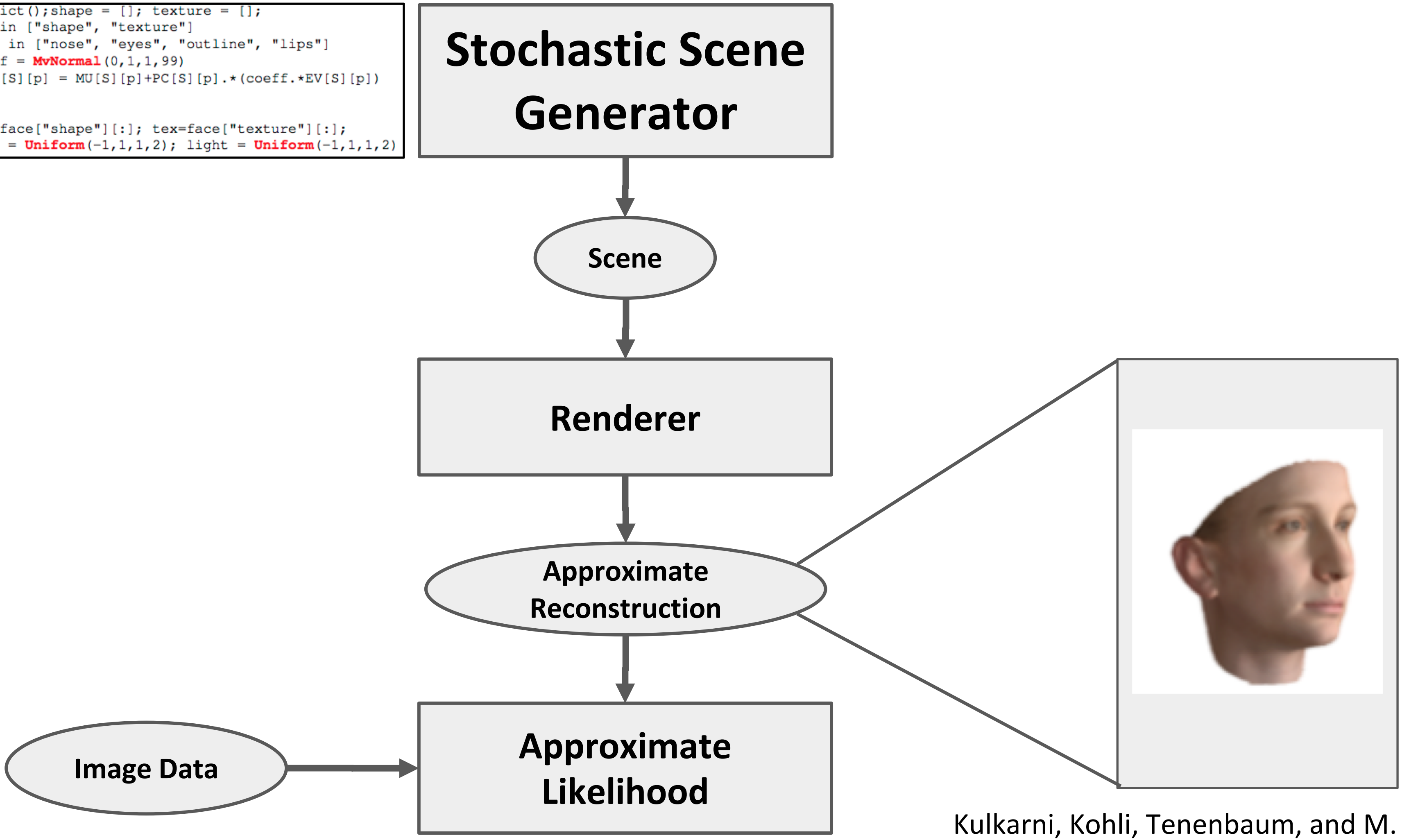
Kulkarni, Kohli, Tenenbaum, and M. (2015)

```
face=Dict();shape = []; texture = [];
for S in ["shape", "texture"]
 for p in ["nose", "eyes", "outline", "lips"]
  coeff = MvNormal(0,1,1,99)
  face[S][p] = MU[S][p]+PC[S][p].*(coeff.*EV[S][p])
 end
end
shape=face["shape"][:]; tex=face["texture"][:];
camera = Uniform(-1,1,1,2); light = Uniform(-1,1,1,2)
```



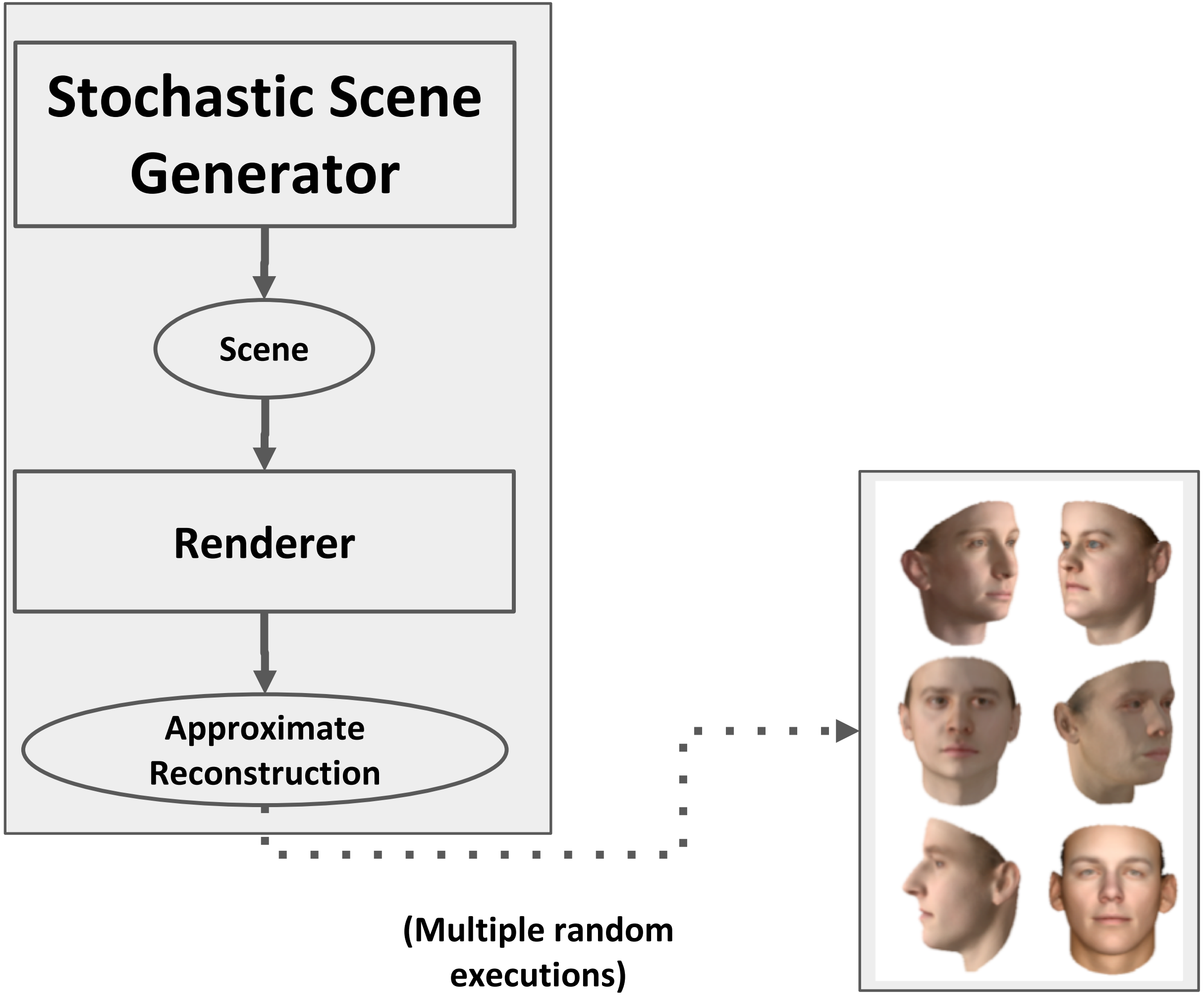
Kulkarni, Kohli, Tenenbaum, and M. (2015)

```
face=Dict();shape = []; texture = [];
for S in ["shape", "texture"]
 for p in ["nose", "eyes", "outline", "lips"]
  coeff = MvNormal(0,1,1,99)
  face[S][p] = MU[S][p]+PC[S][p].*(coeff.*EV[S][p])
 end
end
shape=face["shape"][:]; tex=face["texture"][:];
camera = Uniform(-1,1,1,2); light = Uniform(-1,1,1,2)
```

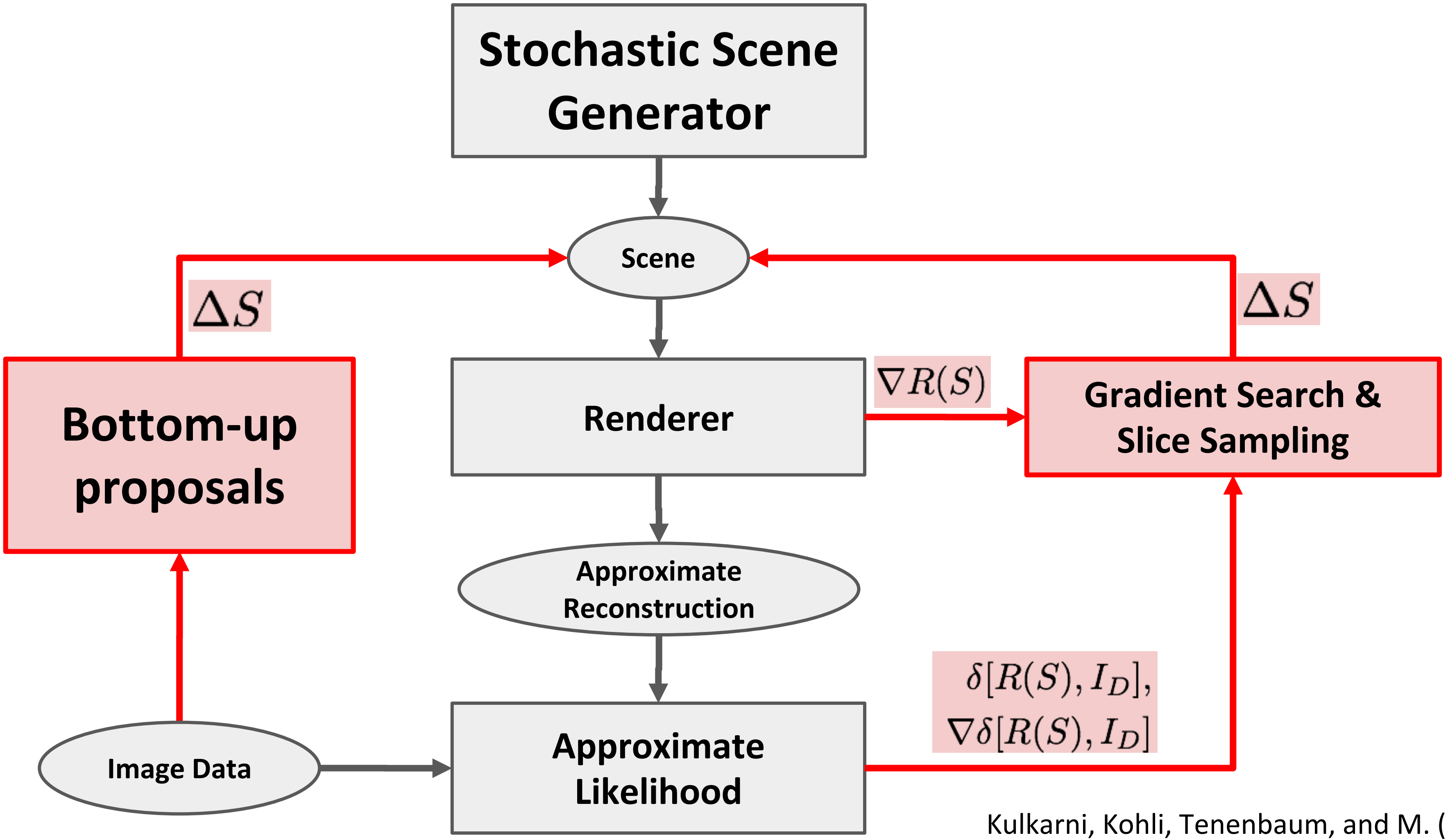


Kulkarni, Kohli, Tenenbaum, and M. (2015)

```
face=Dict();shape = []; texture = [];
for S in ["shape", "texture"]
 for p in ["nose", "eyes", "outline", "lips"]
  coeff = MvNormal(0,1,1,99)
  face[S][p] = MU[S][p]+PC[S][p].*(coeff.*EV[S][p])
 end
end
shape=face["shape"][:]; tex=face["texture"][:];
camera = Uniform(-1,1,1,2); light = Uniform(-1,1,1,2)
```

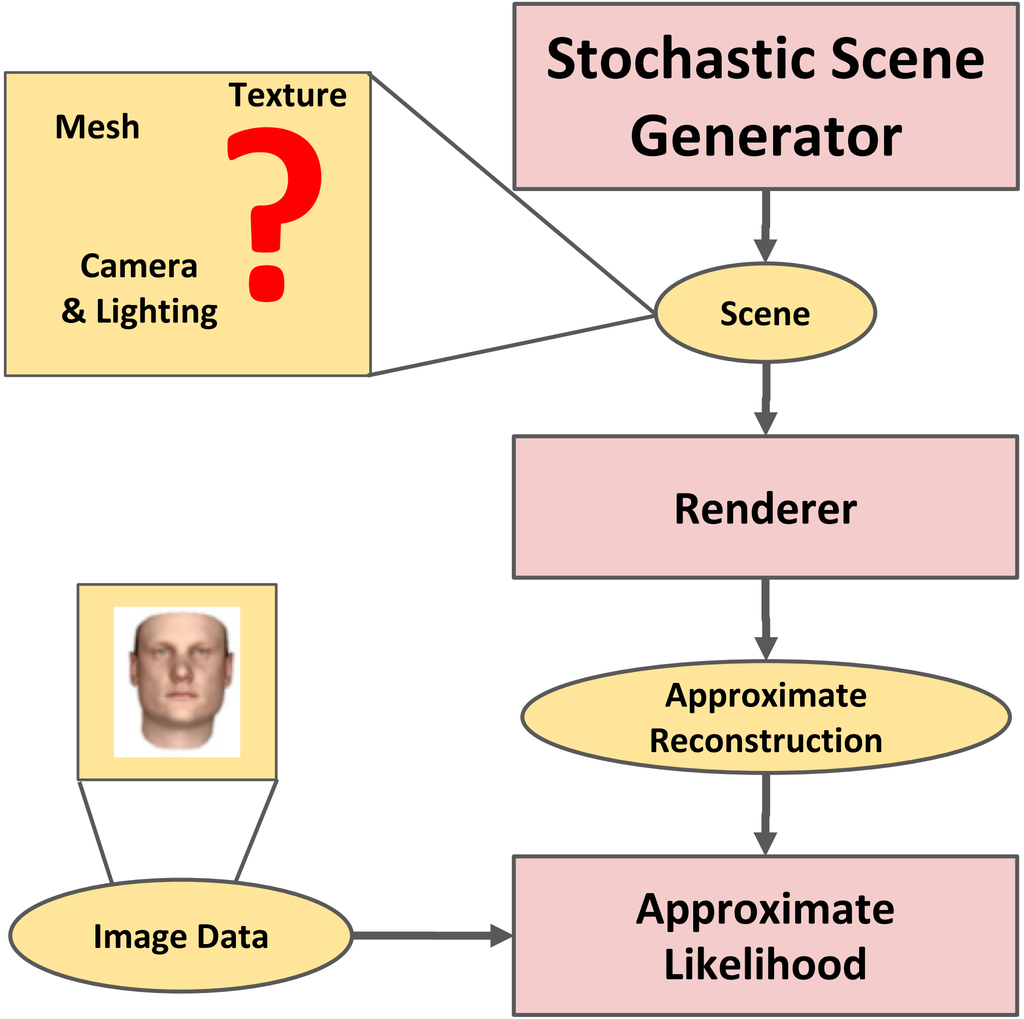


Kulkarni, Kohli, Tenenbaum, and M. (2015)



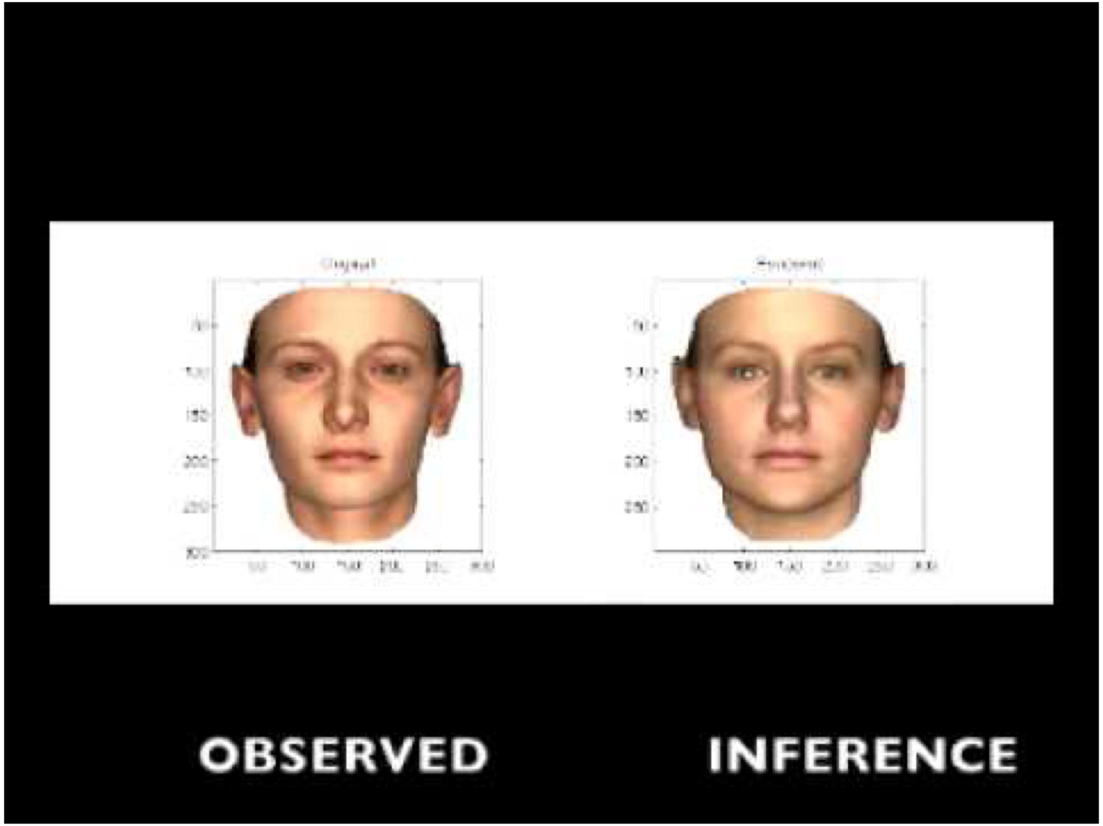
Kulkarni, Kohli, Tenenbaum, and M. (2015)

```
function PROGRAM(MU, PC, EV, VERTEX_ORDER)
  # Scene Language: Stochastic Scene Gen
  face=Dict();shape = []; texture = [];
  for S in ["shape", "texture"]
   for p in ["nose", "eyes", "outline", "lips"]
    coeff = MvNormal(0,1,1,99)
    face[S][p] = MU[S][p]+PC[S][p].*(coeff.*EV[S][p])
   end
  end
  shape=face["shape"][:]; tex=face["texture"][:];
  camera = Uniform(-1,1,1,2); light = Uniform(-1,1,1,2)

  # Approximate Renderer
  rendered_img= MeshRenderer(shape,tex,light,camera)

  # Representation Layer
  ren_ftrs = getFeatures("CNN_Conv6", rendered_img)

  # Comparator
  #Using Pixel as Summary Statistics
  observe(MvNormal(0,0.01), rendered_img-obs_img)
  #Using CNN last conv layer as Summary Statistics
  observe(MvNormal(0,10), ren_ftrs-obs_cnn)
end

global obs_img = imread("test.png")
global obs_cnn = getFeatures("CNN_Conv6", img)
#Load args from file
TR = trace(PROGRAM,args=[MU,PC,EV,VERTEX_ORDER])
# Data-Driven Learning
learn_datadriven_proposals(TR,100000,"CNN_Conv6")
load_proposals(TR)
# Inference
infer(TR,CB,20,["DATA-DRIVEN"])
infer(TR,CB,200,["ELLIPTICAL"])
```



"Find a face shape and texture that matches this input image."

Input Image



Reconstruction

$R(S) = I_R$

Kulkarni, Kohli, Tenenbaum, and Mansinghka (2015)

Gen: a general-purpose probabilistic programming platform with programmable inference

Modeling and inference from multiple paradigms

Bayesian networks, Markov random fields, graphics/physics engines, deep neural network models

Monte Carlo inference, deep inference networks, numerical optimization

Programmable inference, not black-box

"Use Gibbs sampling to update X|Y, then optimize Y|X"

Advanced techniques, e.g. reversible jump and particle MCMC

Custom MCMC/SMC proposals, without requiring users to derive proposal densities and Jacobians

Easy to combine built-in algorithms with arbitrary user-specified inference code

Fast enough for real-time applications

Out-of-the-box performance competitive with handwritten samplers

Users can optimize performance for slow components

Cusumano-Towner et al. (2018)

Example: body pose inference as inverse graphics

```
struct BodyPose
    body_rotation::Point3D
    elbow_right_loc::Point3D
    elbow_left_loc::Point3D
    ...
end
```

3D model

Renderer

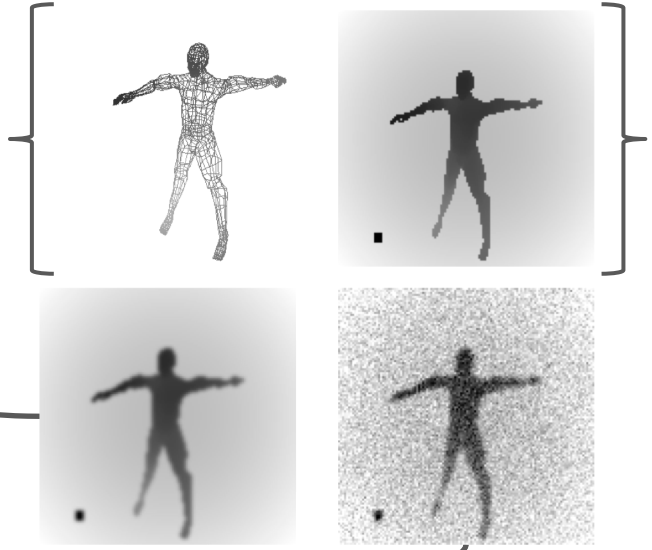
Inference

Cusumano-Towner et al. (2018)

Generative model based on a graphics engine

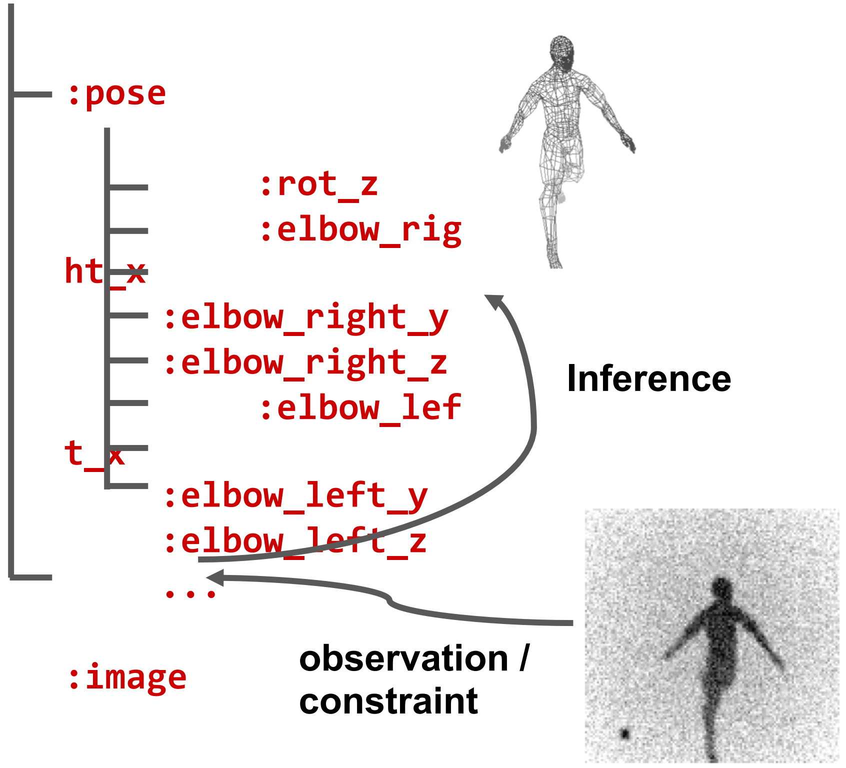
```
@gen function body_pose_prior()
    ...
end

@gen function generative_model()

    # sample pose from prior
    pose = @addr(body_pose_prior(), :pose)

    # render depth image and add blur
    image = render_depth_image(pose)
    blurred = gaussian_blur(image, 1)

    # pixel-wise likelihood model
    @addr(pixel_noise(blurred, 0.1), :image)
end
```

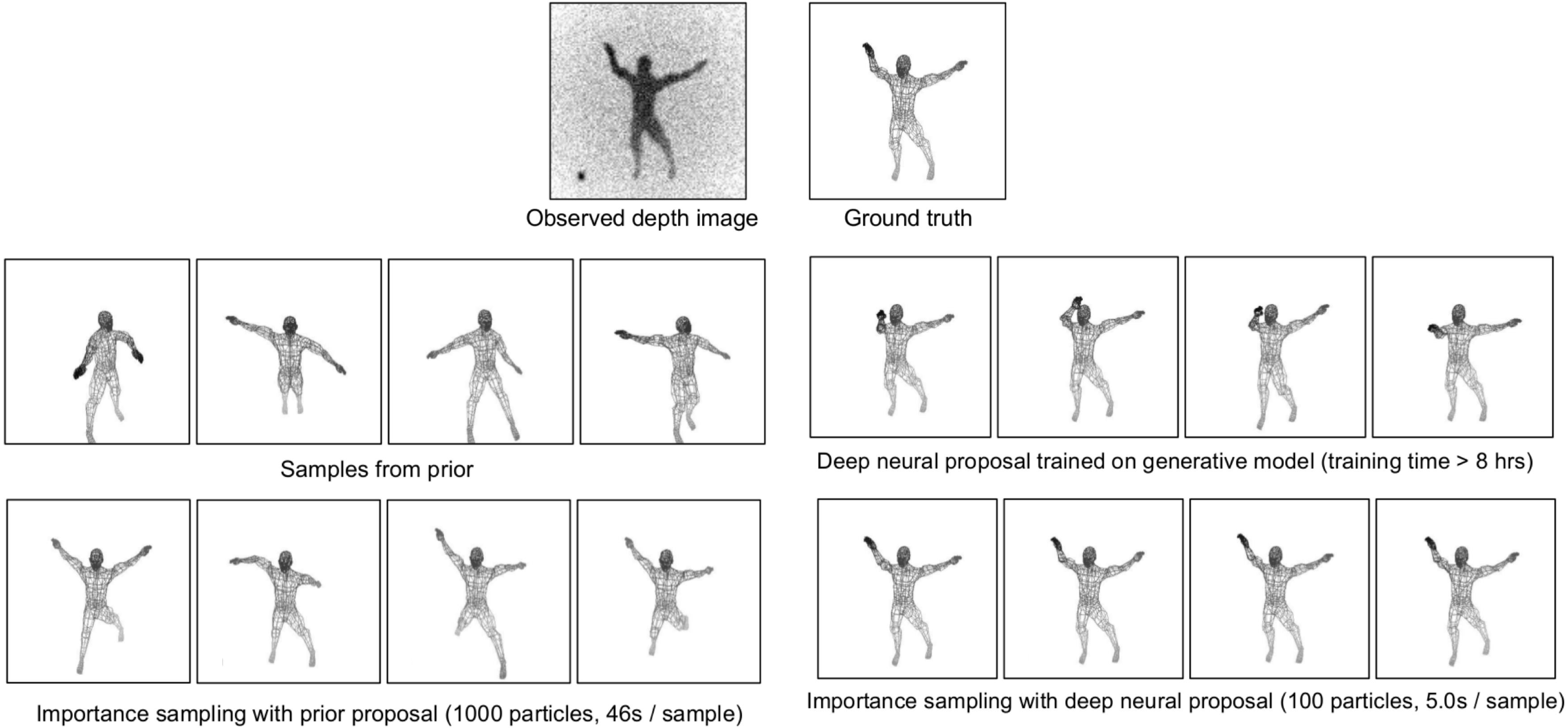
```
struct BodyPose
    rotation::Point3
    elbow_r_loc::Point3
    elbow_l_loc::Point3
    ...
end
```



Generative model based on a graphics engine

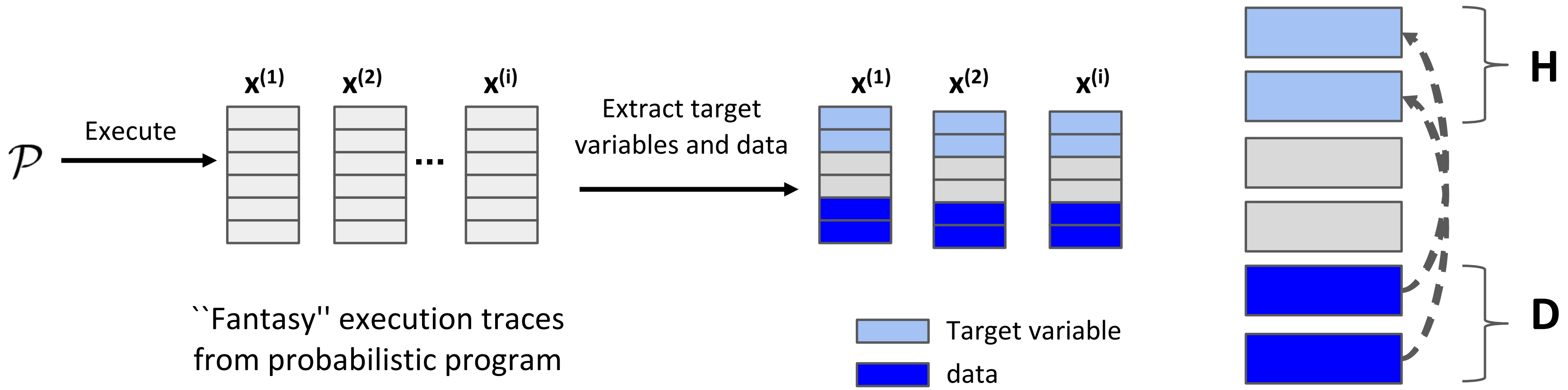
```
@gen function body_pose_prior()
    ...
end

@gen function generative_model()

    # sample pose from prior
    pose = @addr(body_pose_prior(), :pose)

    # render depth image and add blur
    image = render_depth_image(pose)
    blurred = gaussian_blur(image, 1)

    # pixel-wise likelihood model
    @addr(pixel_noise(blurred, 0.1), :image)
end
```



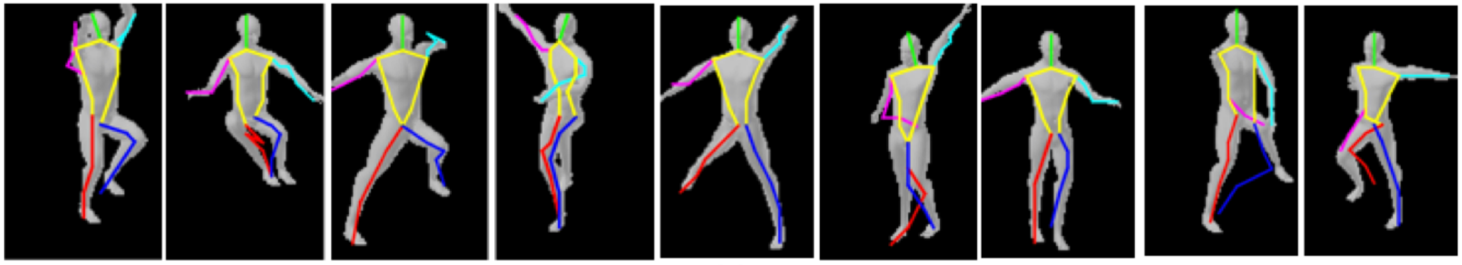


Cusumano-Towner et al. (2018)

Inference using deep learning and Monte Carlo



Examples of ``fantasy'' execution traces including target variables and data



Challenge: integrating multiple modeling & inference paradigms

Monte Carlo in generative models

- Models defined by arbitrary generative code in Julia
- Fast editing of execution traces during MCMC inference, via incremental computation
- Fast resampling of execution traces for SMC inference, via persistent data structures

Deep learning

- Models defined by differentiable TensorFlow computations mixed with Julia code
- Batched gradients with respect to large parameter arrays located on GPU

Gradient-based inference

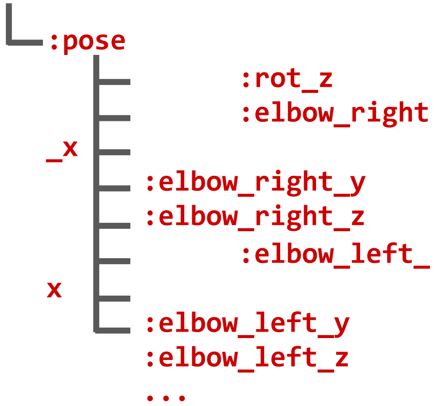
- Gradients with respect to ~10s of random variables (non-contiguous in memory)
- MAP, HMC, MALA, etc.

Cusumano-Towner et al. (2018)

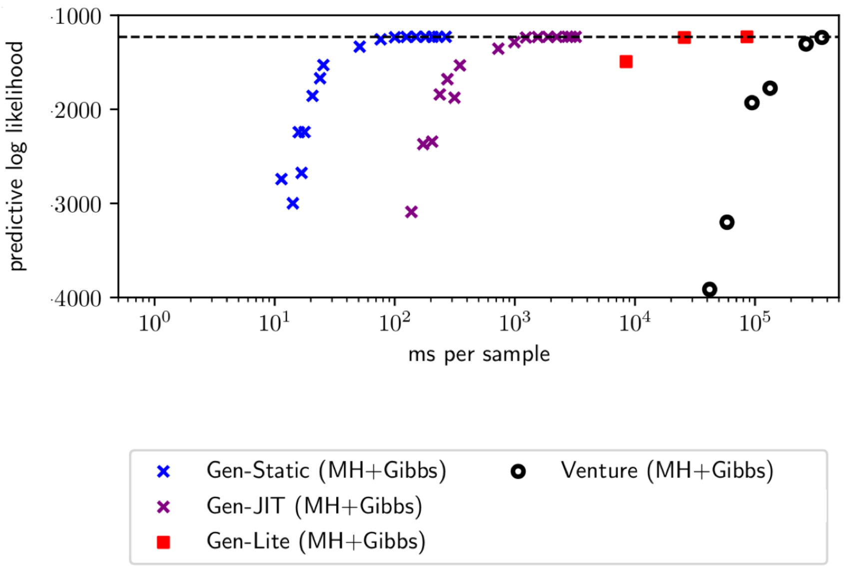
```julia
@gen function neural_proposal_batched(images::Vector{Matrix{Float64}})

    images_flat = vectorize_images(images)

    # run inference network in batch
    output_layer = @addr(neural_network(images_flat), :network)

    # make prediction for each image given inference network outputs
    batch_size = length(images)
    for i=1:batch_size
        @addr(predict_body_pose(outputs[i,:]), :poses => i)
    end
end
```

```
└ :poses
   ├ 1
   │  ├ :rot_z
   │  ├ :elbow_right_x
   │  ├ :elbow_right_y
   │  ├ :elbow_right_z
   │  ├ :elbow_left_x
   │  ├ :elbow_left_y
   │  ├ :elbow_left_z
   │  └ ...
   ├ 2
   │  ├ :rot_z
   │  ├ :elbow_right_x
   │  ├ :elbow_right_y
   │  ├ :elbow_right_z
   │  ├ :elbow_left_x
   │  ├ :elbow_left_y
   │  ├ :elbow_left_z
   │  └ ...
   3
   ...
```

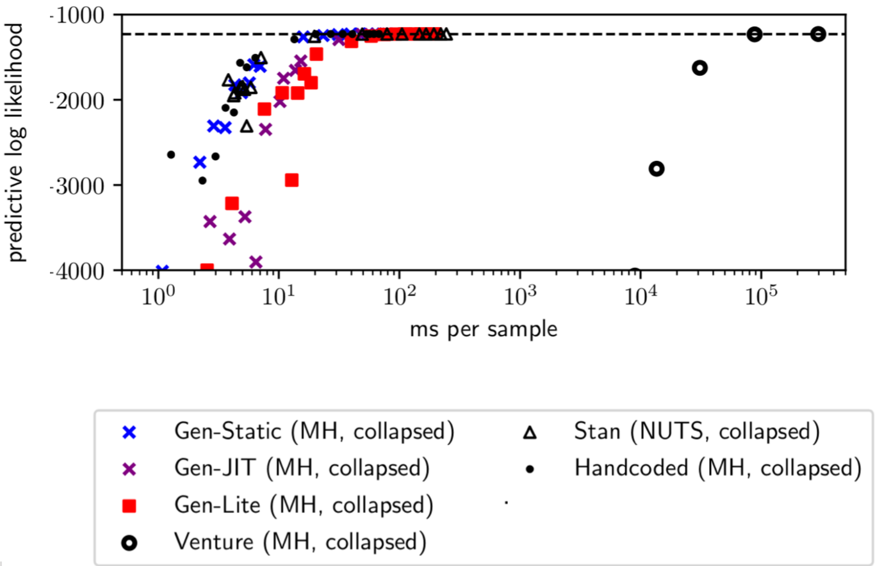
```
@gen function neural_proposal(image::Matrix{Float64})
    image_flat = reshape(image, 1, 128 * 128)
    output_layer = @addr(neural_network(image_flat), :network)
    @addr(predict_body_pose(output_layer[1,:]), :pose)
end

neural_network = @tensorflow_module begin

  @input image_flat Float32 [-1, 128 * 128]
  image = tf.reshape(image_flat, [-1, 128, 128, 1])

  @param W_conv1 initial_weight([5, 5, 1, 32])
  @param b_conv1 initial_bias([32])
  h_conv1 = tf.nn.relu(conv2d(image, W_conv1) + b_conv1)
  h_pool1 = max_pool_2x2(h_conv1)
  ...

  @param W_fc1 initial_weight([16 * 16 * 64, 1024])
  @param b_fc1 initial_bias([1024])
  h_fc1 = tf.nn.relu(h_pool3_flat * W_fc1 + b_fc1)

  @param W_fc2 initial_weight([1024, 32])
  @param b_fc2 initial_bias([32])

  @output Float32 (tf.matmul(h_fc1, W_fc2) + b_fc2)
end
```



Performance of Gen's JIT compiler

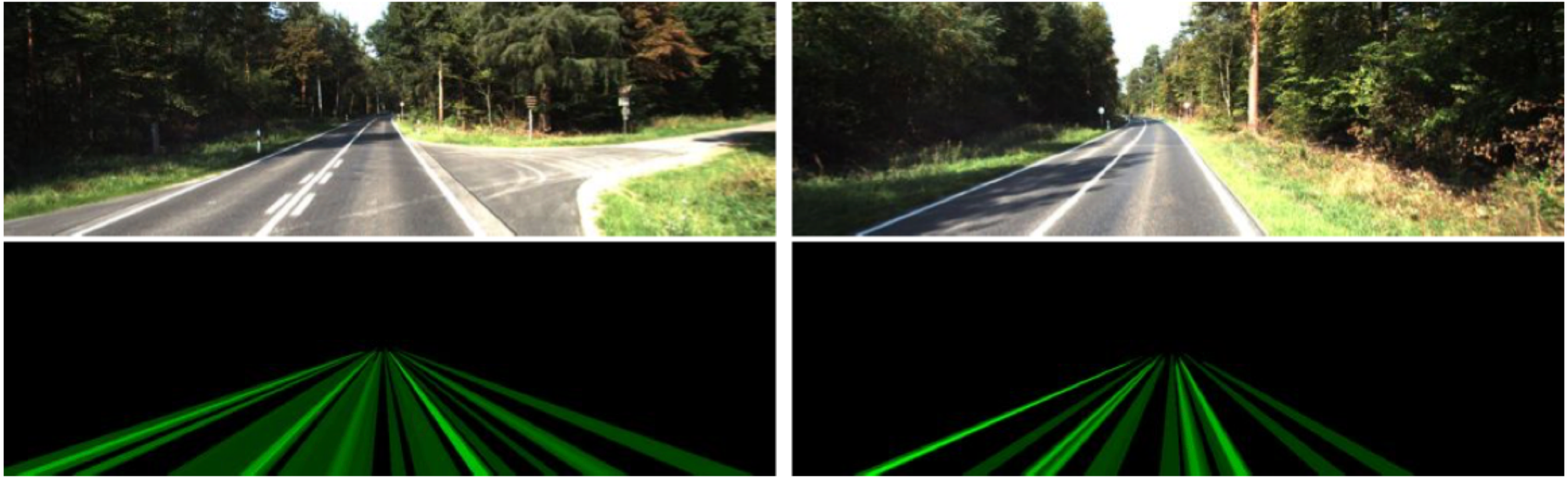


Uncollapsed model



Manually collapsed model

Cusumano-Towner et al. (2018)



High uncertainty due to violated assumptions

Lower uncertainty for unsurprising data

Mansinghka*, Kulkarni*, et al. (2013)

Outline

1. **Motivation**
2. **What is probabilistic programming?**

    Pedagogical example: simple (or not-so-simple) curve fitting

3. **Programmable inference, not just black-box**

    Application: machine perception via inverse graphics

4. **Learning the structure and parameters of probabilistic programs**

    Application: automatic data modeling for scientific data analysis

5. **The MIT Modeling and Inference Stack**

Lab experiment

Experimental data

	strain_name	time_point	temperature	Actuator_YFP	RiboJ00_Part_ribozyme	ybiT	ttdR	metR	cysM	preA
	MG1655_Genomic_IcaR_Gate	18.0	37.0	7182.814030	23196.715220	56.407155	8.122473	3.404433	22.554367	13.138466
	MG1655_Genomic_IcaR_Gate	18.0	37.0	6850.282154	20212.067980	66.983175	1.890360	4.621870	35.776926	7.134732
	MG1655_Genomic_IcaR_Gate	18.0	37.0	6459.667717	12657.394760	105.104475	5.078912	6.622817	56.288495	14.057411
	MG1655_Genomic_IcaR_Gate	18.0	37.0	5384.380877	10816.005350	78.503822	4.467902	6.991284	35.652097	14.164986
	MG1655_Genomic_NAND_Circuit	18.0	37.0	29984.205560	57512.309870	83.724520	15.151475	43.165337	55.936072	14.918031
	MG1655_Genomic_NAND_Circuit	18.0	37.0	34582.809280	87128.520830	101.577667	8.759255	20.559459	48.389122	13.223924
	MG1655_Genomic_NAND_Circuit	18.0	37.0	31519.319620	76236.806450	126.530937	7.274019	23.475866	65.485309	17.021557
	MG1655_Genomic_NAND_Circuit	18.0	37.0	35594.041500	114552.584000	90.227517	4.644846	29.526906	53.853729	8.181092
	MG1655_Genomic_pTACmin	18.0	37.0	1616.725667	3313.110829	66.335180	7.078774	11.249797	29.872311	15.362400
	MG1655_Genomic_pTACmin	18.0	37.0	1913.662092	4027.111166	82.239438	9.683810	15.389797	49.533963	11.878533

Virtual experiment simulator, as probabilistic program

```
(define generate-virtual-experimental-results-using-model-1
  (gen []

    (define cluster-for-actuator_yfp-and-riboj00_part_ribozyme (
        categorical [0.62 0.29 0.09]))

    (define [actuator_yfp-mean actuator_yfp-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [34278.55 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [336058.53125 432304.475202]))
    (define actuator_yfp (gaussian actuator_yfp-mean actuator_yfp-std))

    (define [riboj00_part_ribozyme-mean riboj00_part_ribozyme-std] (cond

■ ■ ■
```

Use cases for probabilistic programs that model a virtual experiment

1. Screen new batches of data for ETL errors and lab protocol execution errors

2. Detect drift between old and new batches of data

3. Detect multivariate relationships among experimental variables, and quantify their probable strength

4. Estimate anticipated variability in outcome for a given experimental condition

Hard to write

Experimental data

strain_name	time_point	temperature	Actuator_YFP	RiboJ00_Part_ribozyme	ybiT	ttdR	metR	cysM	preA
MG1655_Genomic_IcaR_Gate	18.0	37.0	7182.814030	23196.715220	56.407155	8.122473	3.404433	22.554367	13.138466
MG1655_Genomic_IcaR_Gate	18.0	37.0	6850.282154	20212.067980	66.983175	1.890360	4.621870	35.776926	7.134732
MG1655_Genomic_IcaR_Gate	18.0	37.0	6459.667717	12657.394760	105.104475	5.078912	6.622817	56.288495	14.057411
MG1655_Genomic_IcaR_Gate	18.0	37.0	5384.380877	10816.005350	78.503822	4.467902	6.991284	35.652097	14.164986
MG1655_Genomic_NAND_Circuit	18.0	37.0	29984.205560	57512.309870	83.724520	15.151475	43.165337	55.936072	14.918031
MG1655_Genomic_NAND_Circuit	18.0	37.0	34582.809280	87128.520830	101.577667	8.759255	20.559459	48.389122	13.223924
MG1655_Genomic_NAND_Circuit	18.0	37.0	31519.319620	76236.806450	126.530937	7.274019	23.475866	65.485309	17.021557
MG1655_Genomic_NAND_Circuit	18.0	37.0	35594.041500	114552.584000	90.227517	4.644846	29.526906	53.853729	8.181092
MG1655_Genomic_pTACmin	18.0	37.0	1616.725667	3313.110829	66.335180	7.078774	11.249797	29.872311	15.362400
MG1655_Genomic_pTACmin	18.0	37.0	1913.662092	4027.111166	82.239438	9.683810	15.389797	49.533963	11.878533

Virtual experiment simulator, as probabilistic program

```
(define generate-virtual-experimental-results-using-model-1
  (gen []

    (define cluster-for-actuator_yfp-and-riboj00_part_ribozyme (
        categorical [0.62 0.29 0.09]))

    (define [actuator_yfp-mean actuator_yfp-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [34278.55 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [336058.53125 432304.475202]))
    (define actuator_yfp (gaussian actuator_yfp-mean actuator_yfp-std))

    (define [riboj00_part_ribozyme-mean riboj00_part_ribozyme-std] (cond

■ ■ ■
```

Lab experiment

… but relevant data is often available

Experimental data

strain_name	time_point	temperature	Actuator_YFP	RiboJ00_Part_ribozyme	ybiT	ttdR	metR	cysM	preA
MG1655_Genomic_IcaR_Gate	18.0	37.0	7182.814030	23196.715220	56.407155	8.122473	3.404433	22.554367	13.138466
MG1655_Genomic_IcaR_Gate	18.0	37.0	6850.282154	20212.067980	66.983175	1.890360	4.621870	35.776926	7.134732
MG1655_Genomic_IcaR_Gate	18.0	37.0	6459.667717	12657.394760	105.104475	5.078912	6.622817	56.288495	14.057411
MG1655_Genomic_IcaR_Gate	18.0	37.0	5384.380877	10816.005350	78.503822	4.467902	6.991284	35.652097	14.164986
MG1655_Genomic_NAND_Circuit	18.0	37.0	29984.205560	57512.309870	83.724520	15.151475	43.165337	55.936072	14.918031
MG1655_Genomic_NAND_Circuit	18.0	37.0	34582.809280	87128.520830	101.577667	8.759255	20.559459	48.389122	13.223924
MG1655_Genomic_NAND_Circuit	18.0	37.0	31519.319620	76236.806450	126.530937	7.274019	23.475866	65.485309	17.021557
MG1655_Genomic_NAND_Circuit	18.0	37.0	35594.041500	114552.584000	90.227517	4.644846	29.526906	53.853729	8.181092
MG1655_Genomic_pTACmin	18.0	37.0	1616.725667	3313.110829	66.335180	7.078774	11.249797	29.872311	15.362400
MG1655_Genomic_pTACmin	18.0	37.0	1913.662092	4027.111166	82.239438	9.683810	15.389797	49.533963	11.878533

Virtual experiment simulator, as probabilistic program

```
(define generate-virtual-experimental-results-using-model-1
  (gen []

    (define cluster-for-actuator_yfp-and-riboj00_part_ribozyme (
        categorical [0.62 0.29 0.09]))

    (define [actuator_yfp-mean actuator_yfp-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [34278.55 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [336058.53125 432304.475202]))
    (define actuator_yfp (gaussian actuator_yfp-mean actuator_yfp-std))

    (define [riboj00_part_ribozyme-mean riboj00_part_ribozyme-std] (cond

                         ■ ■ ■
```

Can we automatically build probabilistic programs that model the data?

Experimental data

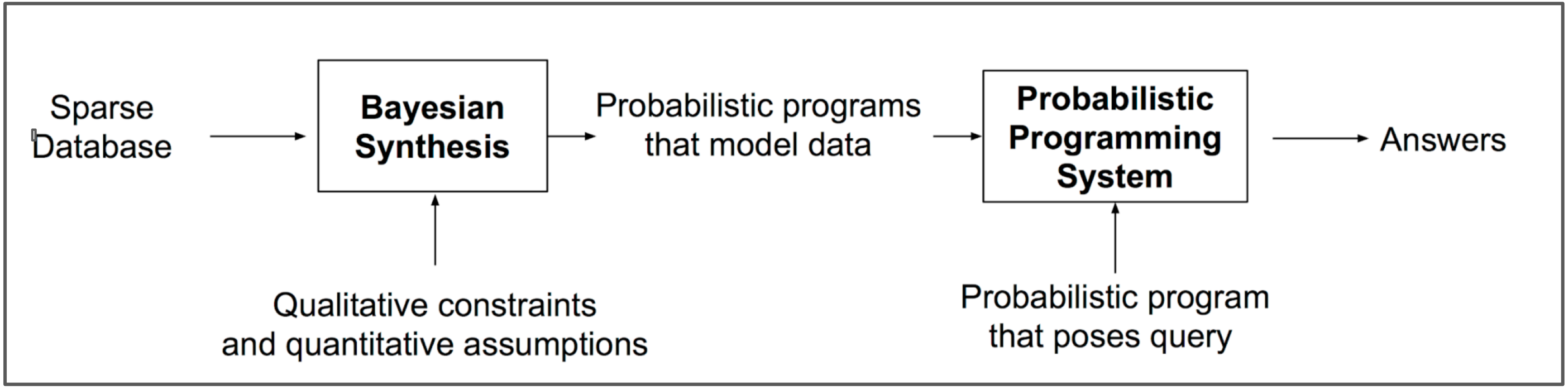
strain_name	time_point	temperature	Actuator_YFP	RiboJ00_Part_ribozyme	ybiT	ttdR	metR	cysM	preA
MG1655_Genomic_IcaR_Gate	18.0	37.0	7182.814030	23196.715220	56.407155	8.122473	3.404433	22.554367	13.138466
MG1655_Genomic_IcaR_Gate	18.0	37.0	6850.282154	20212.067980	66.983175	1.890360	4.621870	35.776926	7.134732
MG1655_Genomic_IcaR_Gate	18.0	37.0	6459.667717	12657.394760	105.104475	5.078912	6.622817	56.288495	14.057411
MG1655_Genomic_IcaR_Gate	18.0	37.0	5384.380877	10816.005350	78.503822	4.467902	6.991284	35.652097	14.164986
MG1655_Genomic_NAND_Circuit	18.0	37.0	29984.205560	57512.309870	83.724520	15.151475	43.165337	55.936072	14.918031
MG1655_Genomic_NAND_Circuit	18.0	37.0	34582.809280	87128.520830	101.577667	8.759255	20.559459	48.389122	13.223924
MG1655_Genomic_NAND_Circuit	18.0	37.0	31519.319620	76236.806450	126.530937	7.274019	23.475866	65.485309	17.021557
MG1655_Genomic_NAND_Circuit	18.0	37.0	35594.041500	114552.584000	90.227517	4.644846	29.526906	53.853729	8.181092
MG1655_Genomic_pTAC[illegible]	[illegible]	[illegible]	[illegible]16.725667	3313.110829	66.335180	7.078774	11.249797	29.872311	15.362400
MG1655_Genomic_pTAC[illegible]	[illegible]	[illegible]	[illegible]13.662092	4027.111166	82.239438	9.683810	15.389797	49.533963	11.878533

?

Virtual experiment simulator, as probabilistic program

```
(define generate-virtual-experimental-results-using-model-1
  (gen []

    (define cluster-for-actuator_yfp-and-riboj00_part_ribozyme (
        categorical [0.62 0.29 0.09]))

    (define [actuator_yfp-mean actuator_yfp-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [34278.55 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [336058.53125 432304.475202]))
    (define actuator_yfp (gaussian actuator_yfp-mean actuator_yfp-std))

    (define [riboj00_part_ribozyme-mean riboj00_part_ribozyme-std] (cond

...
```

Automated data modeling for science via Bayesian probabilistic program synthesis



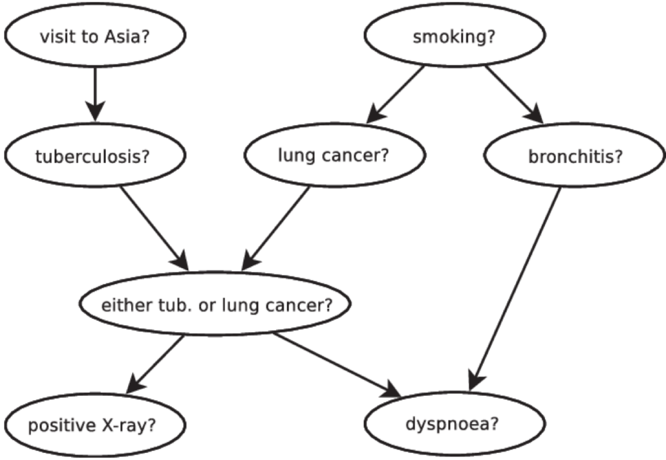
Mansinghka et al. (arXiv 2015)
Mansinghka et al. (JMLR 2016)
Saad & Mansinghka (NIPS 2016)
Saad & Mansinghka (AISTATS 2017)
Saad & Schaechtle et al. (under review; arXiv 2017)

⚠ Technical challenge: structure learning is hard...

Robust automatic data modeling requires learning model structure, not just parameters...

Remember Bayesian network structure learning:

- Search over structures was slow and unreliable
- Hard to include hidden variables, leading to underfitting
- Hard to apply to mixed numerical and discrete data
- Hard to get uncertainty over model structure



… but we can use tools from nonparametric Bayes!

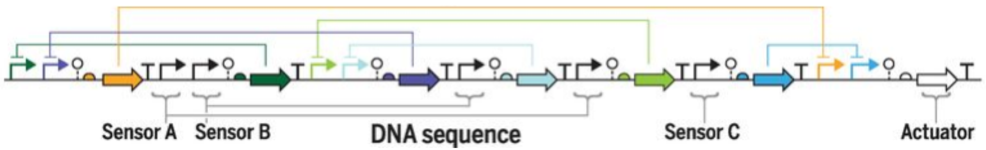
10 years of research & engineering towards CrossCat, a nonparametric Bayesian prior over probabilistic model structure and parameters.

Monte Carlo implementation scales to tables with ~100K rows and ~1K columns

$$
\begin{aligned}
\alpha_D &\sim \mathrm{Gamma}(k=1, \theta=1) && \\
\vec{\lambda}_d &\sim V_d(\cdot) && \text{foreach } d \in \{1,\cdots,D\} \\
z_d &\sim \mathrm{CRP}(\{z_i \mid i \neq d\}; \alpha_D) && \text{foreach } d \in \{1,\cdots,D\} \\
\alpha_v &\sim \mathrm{Gamma}(k=1, \theta=1) && \text{foreach } v \in \vec{z} \\
y_r^v &\sim \mathrm{CRP}(\{y_i^v \mid i \neq r\}; \alpha_v) && \text{foreach } v \in \vec{z} \text{ and} \\
& && r \in \{1,\cdots,R\} \\
\vec{\theta}_c^d &\sim M_d(\cdot; \vec{\lambda}_d) && \text{foreach } v \in \vec{z}, c \in \vec{y}^v\text{, and } d \text{ such that} \\
& && z_d = v \text{ and } u_d = 1 \\
\vec{x}_{(\cdot,d)}^c = \{x_{(r,d)} \mid y_r^{z_d} = c\} &\sim \begin{cases} \prod_r L_d(\vec{\theta}_c^d) & \text{if } u_d = 1 \\ ML_d(\vec{\lambda}_d) & \text{if } u_d = 0 \end{cases} && \text{foreach } v \in \vec{z} \text{ and each } c \in \vec{y}^v
\end{aligned}
$$

Mansinghka et al. (JMLR 2016; NIPS 2009; CogSci 2006); Obermeyer et al. (AISTATS; 2014);

Example dataset from genetic circuit design



Experimental condition

Circuit output (fpkm)

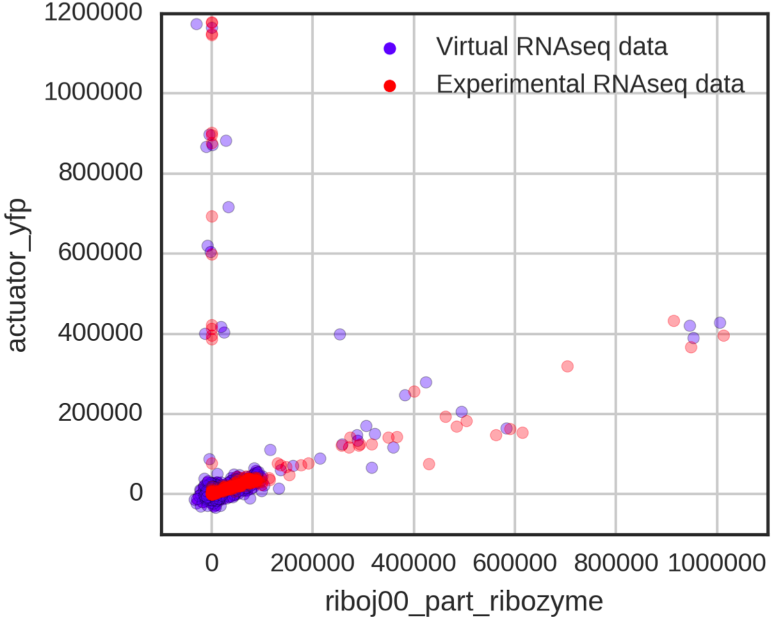
One gene contained in the circuit

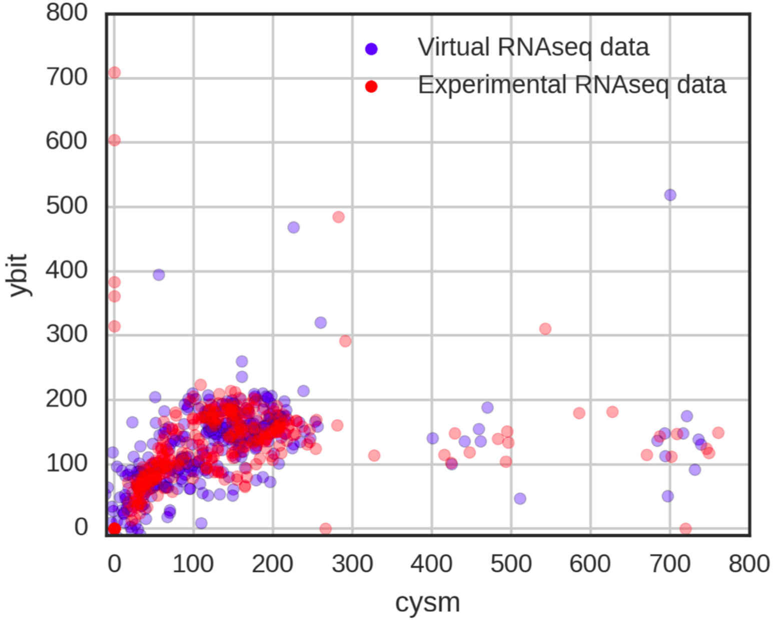
Genes that are not part of the circuit

~320 measurements

strain_name	time_point	temperature	Actuator_YFP	RiboJ00_Part_ribozyme	ybiT	ttdR	metR	cysM	preA
MG1655_Genomic_IcaR_Gate	18.0	37.0	7182.814030	23196.715220	56.407155	8.122473	3.404433	22.554367	13.138466
MG1655_Genomic_IcaR_Gate	18.0	37.0	6850.282154	20212.067980	66.983175	1.890360	4.621870	35.776926	7.134732
MG1655_Genomic_IcaR_Gate	18.0	37.0	6459.667717	12657.394760	105.104475	5.078912	6.622817	56.288495	14.057411
MG1655_Genomic_IcaR_Gate	18.0	37.0	5384.380877	10816.005350	78.503822	4.467902	6.991284	35.652097	14.164986
MG1655_Genomic_NAND_Circuit	18.0	37.0	29984.205560	57512.309870	83.724520	15.151475	43.165337	55.936072	14.918031
MG1655_Genomic_NAND_Circuit	18.0	37.0	34582.809280	87128.520830	101.577667	8.759255	20.559459	48.389122	13.223924
MG1655_Genomic_NAND_Circuit	18.0	37.0	31519.319620	76236.806450	126.530937	7.274019	23.475866	65.485309	17.021557
MG1655_Genomic_NAND_Circuit	18.0	37.0	35594.041500	114552.584000	90.227517	4.644846	29.526906	53.853729	8.181092
MG1655_Genomic_pTACmin	18.0	37.0	1616.725667	3313.110829	66.335180	7.078774	11.249797	29.872311	15.362400
MG1655_Genomic_pTACmin	18.0	37.0	1913.662092	4027.111166	82.239438	9.683810	15.389797	49.533963	11.878533

Compare virtual and experimental RNAseq data






```
%%bql
SIMULATE riboj00_part_ribozyme, actuator_yfp
      FROM data;
```

```
%%bql
SELECT riboj00_part_ribozyme, actuator_yfp
      FROM data;
```

```
%%bql
SIMULATE cysm, ybit FROM data;
```

```
%%bql
SELECT cysm, ybit FROM data;
```

Synthesis with BQL and python

In less than 20 lines of code, we generate probabilistic programs to model a new dataset.

```
%%bql
CREATE TABLE "data_subset" AS
    SELECT
        "Actuator_yfp",
        "riboj00_part_ribozyme",
        "Ybit",
        "cysM" FROM "data”

CREATE POPULATION FOR "data_subset" WITH SCHEMA (
     SET STATTYPES OF
         "Actuator_YFP",
         "riboj00_part_ribozyme",
         "ybiT",
         "cysM" TO NUMERICAL);

CREATE GENERATOR FOR "data_subset";
INITIALIZE 100 MODELS;
ANALYZE "data_subset" FOR 50 ITERATIONS;
```

```
%%python
code = export_to_metaprob("data_subset")
```

Synthesis with BQL and python

For this demo, we use a subset of all the data available, namely:

1. **A part of the circuit and YFP; and**
2. **Two genes that weren't part of the circuit and should not have interactions with it YFP.**

```
%%bql
CREATE TABLE "data_subset" AS
    SELECT
        "Actuator_yfp",
        "riboj00_part_ribozyme",
        "Ybit",
        "cysM" FROM "data"

CREATE POPULATION FOR "data_subset" WITH SCHEMA (
     SET STATTYPES OF
         "Actuator_YFP",
         "riboj00_part_ribozyme",
         "ybiT",
         "cysM" TO NUMERICAL);

CREATE GENERATOR FOR "data_subset";
INITIALIZE 100 MODELS;
ANALYZE "data_subset" FOR 50 ITERATIONS;
```

```
%%python
code = export_to_metaprob("data_subset")
```

Synthesis with BQL and python

We create a statistical population for this data

```
%%bql
CREATE TABLE "data_subset" AS
    SELECT
        "Actuator_yfp",
        "riboj00_part_ribozyme",
        "Ybit",
        "cysM" FROM "data”

CREATE POPULATION FOR "data_subset" WITH SCHEMA (
    SET STATTYPES OF
        "Actuator_YFP",
        "riboj00_part_ribozyme",
        "ybiT",
        "cysM" TO NUMERICAL);

CREATE GENERATOR FOR "data_subset";
INITIALIZE 100 MODELS;
ANALYZE "data_subset" FOR 50 ITERATIONS;
```

```
%%python
code = export_to_metaprob("data_subset")
```

Synthesis with BQL and python

Run analysis on an ensemble of 100 models

```
%%bql
CREATE TABLE "data_subset" AS
    SELECT
        "Actuator_yfp",
        "riboj00_part_ribozyme",
        "Ybit",
        "cysM" FROM "data”

CREATE POPULATION FOR "data_subset" WITH SCHEMA (
     SET STATTYPES OF
         "Actuator_YFP",
         "riboj00_part_ribozyme",
         "ybiT",
         "cysM" TO NUMERICAL);

CREATE GENERATOR FOR "data_subset";
INITIALIZE 100 MODELS;
ANALYZE "data_subset" FOR 50 ITERATIONS;

%%python
code = export_to_metaprob("data_subset")
```

Synthesis with BQL and python

```
%%bql
CREATE TABLE "data_subset" AS
    SELECT
        "Actuator_yfp",
        "riboj00_part_ribozyme",
        "Ybit",
        "cysM" FROM "data”

CREATE POPULATION FOR "data_subset" WITH SCHEMA (
     SET STATTYPES OF
         "Actuator_YFP",
         "riboj00_part_ribozyme",
         "ybiT",
         "cysM" TO NUMERICAL);

CREATE GENERATOR FOR "data_subset";
INITIALIZE 100 MODELS;
ANALYZE "data_subset" FOR 50 ITERATIONS;
```

Export the learned ensemble of models to Metaprob

```
%%python
code = export_to_metaprob("data_subset")
```

The result of synthesis:

- executable with Metaprob;
- human readable; and
- editable.

```
(define generate-virtual-experimental-results-using-model-1
  (gen []

    (define cluster-for-actuator_yfp-and-riboj00_part_ribozyme (
        categorical [0.62 0.29 0.09]))

    (define [actuator_yfp-mean actuator_yfp-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [34278.55 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [336058.53125 432304.475202]))
    (define actuator_yfp (gaussian actuator_yfp-mean actuator_yfp-std))

    (define [riboj00_part_ribozyme-mean riboj00_part_ribozyme-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [83284.60 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [0.0 0.01]))
    (define riboj00_part_ribozyme
      (gaussian riboj00_part_ribozyme-mean riboj00_part_ribozyme-std))

    (define cluster-for-ybit-and-cysm (categorical [0.59 0.24 0.09 0.05 0.02 0.01]))

    (define [ybit-mean ybit-std]  (cond
      (= cluster-for-ybit-and-cysm 0) [149.01 46.92]
      (= cluster-for-ybit-and-cysm 1) [72.21 15.56]
      (= cluster-for-ybit-and-cysm 2) [185.27 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [762.128173828 0.01]
      (= cluster-for-ybit-and-cysm 5) [0.0 0.01]))
    (define ybit (gaussian ybit-mean ybit-std))

    (define [cysm-mean cysm-std] (cond
      (= cluster-for-ybit-and-cysm 0) [150.76 46.92]
      (= cluster-for-ybit-and-cysm 1) [43.56 15.56]
      (= cluster-for-ybit-and-cysm 2) [641.10 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 5) [813.07 420.35]))
    (define cysm (gaussian cysm-mean cysm-std))

    (define virtual-experimental-results [actuator_yfp riboj00_part_ribozyme ybit cysm])
    virtual-experimental-results))
```

The bql code above learned an ensemble of 100 models. We inspect the code for one of them (model #1).

```
(define generate-virtual-experimental-results-using-model-1
  (gen []

    (define cluster-for-actuator_yfp-and-riboj00_part_ribozyme (
        categorical [0.62 0.29 0.09]))

    (define [actuator_yfp-mean actuator_yfp-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [34278.55 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [336058.53125 432304.475202]))
    (define actuator_yfp (gaussian actuator_yfp-mean actuator_yfp-std))

    (define [riboj00_part_ribozyme-mean riboj00_part_ribozyme-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [83284.60 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [0.0 0.01]))
    (define riboj00_part_ribozyme
        (gaussian riboj00_part_ribozyme-mean riboj00_part_ribozyme-std))

    (define cluster-for-ybit-and-cysm (categorical [0.59 0.24 0.09 0.05 0.02 0.01]))

    (define [ybit-mean ybit-std]  (cond
      (= cluster-for-ybit-and-cysm 0) [149.01 46.92]
      (= cluster-for-ybit-and-cysm 1) [72.21 15.56]
      (= cluster-for-ybit-and-cysm 2) [185.27 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [762.128173828 0.01]
      (= cluster-for-ybit-and-cysm 5) [0.0 0.01]))
    (define ybit (gaussian ybit-mean ybit-std))

    (define [cysm-mean cysm-std] (cond
      (= cluster-for-ybit-and-cysm 0) [150.76 46.92]
      (= cluster-for-ybit-and-cysm 1) [43.56 15.56]
      (= cluster-for-ybit-and-cysm 2) [641.10 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 5) [813.07 420.35]))
    (define cysm (gaussian cysm-mean cysm-std))

    (define virtual-experimental-results [actuator_yfp riboj00_part_ribozyme ybit cysm])
    virtual-experimental-results))
```

The learned model indicates that actuator_yfp and riboj00_part_ribozyme are dependent variables.

This implies that for those two variables, we synthesized a 2-d Gaussian mixture model.

To sample new values for actuator_yfp and riboj00_part_ribozyme, we first need to sample the mixture component (cluster).

```
(define generate-virtual-experimental-results-using-model-1
  (gen []

    (define cluster-for-actuator_yfp-and-riboj00_part_ribozyme (
        categorical [0.62 0.29 0.09]))

    (define [actuator_yfp-mean actuator_yfp-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [34278.55 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [336058.53125 432304.475202]))
    (define actuator_yfp (gaussian actuator_yfp-mean actuator_yfp-std))

    (define [riboj00_part_ribozyme-mean riboj00_part_ribozyme-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [83284.60 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [0.0 0.01]))
    (define riboj00_part_ribozyme
      (gaussian riboj00_part_ribozyme-mean riboj00_part_ribozyme-std))

    (define cluster-for-ybit-and-cysm (categorical [0.59 0.24 0.09 0.05 0.02 0.01]))

    (define [ybit-mean ybit-std]  (cond
      (= cluster-for-ybit-and-cysm 0) [149.01 46.92]
      (= cluster-for-ybit-and-cysm 1) [72.21 15.56]
      (= cluster-for-ybit-and-cysm 2) [185.27 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [762.128173828 0.01]
      (= cluster-for-ybit-and-cysm 5) [0.0 0.01]))
    (define ybit (gaussian ybit-mean ybit-std))

    (define [cysm-mean cysm-std] (cond
      (= cluster-for-ybit-and-cysm 0) [150.76 46.92]
      (= cluster-for-ybit-and-cysm 1) [43.56 15.56]
      (= cluster-for-ybit-and-cysm 2) [641.10 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 5) [813.07 420.35]))
    (define cysm (gaussian cysm-mean cysm-std))

    (define virtual-experimental-results [actuator_yfp riboj00_part_ribozyme ybit cysm])
    virtual-experimental-results))
```

The parametrization, i.e. mean and standard deviation (std) of the Gaussian components for the mixture model for actuator_yfp depends on the previously sampled cluster id, (cluster-for-actuator_yfp -and-ribojo00-part_ribyzmye).

```
(define generate-virtual-experimental-results-using-model-1
  (gen []

    (define cluster-for-actuator_yfp-and-riboj00_part_ribozyme (
        categorical [0.62 0.29 0.09]))

    (define [actuator_yfp-mean actuator_yfp-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [34278.55 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [336058.53125 432304.475202]))
    (define actuator_yfp (gaussian actuator_yfp-mean actuator_yfp-std))

    (define [riboj00_part_ribozyme-mean riboj00_part_ribozyme-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [83284.60 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [0.0 0.01]))
    (define riboj00_part_ribozyme
        (gaussian riboj00_part_ribozyme-mean riboj00_part_ribozyme-std))

    (define cluster-for-ybit-and-cysm (categorical [0.59 0.24 0.09 0.05 0.02 0.01]))

    (define [ybit-mean ybit-std]  (cond
      (= cluster-for-ybit-and-cysm 0) [149.01 46.92]
      (= cluster-for-ybit-and-cysm 1) [72.21 15.56]
      (= cluster-for-ybit-and-cysm 2) [185.27 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [762.128173828 0.01]
      (= cluster-for-ybit-and-cysm 5) [0.0 0.01]))
    (define ybit (gaussian ybit-mean ybit-std))

    (define [cysm-mean cysm-std] (cond
      (= cluster-for-ybit-and-cysm 0) [150.76 46.92]
      (= cluster-for-ybit-and-cysm 1) [43.56 15.56]
      (= cluster-for-ybit-and-cysm 2) [641.10 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 5) [813.07 420.35]))
    (define cysm (gaussian cysm-mean cysm-std))

    (define virtual-experimental-results [actuator_yfp riboj00_part_ribozyme ybit cysm])
    virtual-experimental-results))
```

We now sample a value for actuator_yfp from a Gaussian with the previously determined mean and standard deviation.

```
(define generate-virtual-experimental-results-using-model-1
  (gen []

    (define cluster-for-actuator_yfp-and-riboj00_part_ribozyme (
        categorical [0.62 0.29 0.09]))

    (define [actuator_yfp-mean actuator_yfp-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [34278.55 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [336058.53125 432304.475202]))
    (define actuator_yfp (gaussian actuator_yfp-mean actuator_yfp-std))

    (define [riboj00_part_ribozyme-mean riboj00_part_ribozyme-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [83284.60 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [0.0 0.01]))
    (define riboj00_part_ribozyme
        (gaussian riboj00_part_ribozyme-mean riboj00_part_ribozyme-std))

    (define cluster-for-ybit-and-cysm (categorical [0.59 0.24 0.09 0.05 0.02 0.01]))

    (define [ybit-mean ybit-std]  (cond
      (= cluster-for-ybit-and-cysm 0) [149.01 46.92]
      (= cluster-for-ybit-and-cysm 1) [72.21 15.56]
      (= cluster-for-ybit-and-cysm 2) [185.27 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [762.128173828 0.01]
      (= cluster-for-ybit-and-cysm 5) [0.0 0.01]))
    (define ybit (gaussian ybit-mean ybit-std))

    (define [cysm-mean cysm-std] (cond
      (= cluster-for-ybit-and-cysm 0) [150.76 46.92]
      (= cluster-for-ybit-and-cysm 1) [43.56 15.56]
      (= cluster-for-ybit-and-cysm 2) [641.10 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 5) [813.07 420.35]))
    (define cysm (gaussian cysm-mean cysm-std))

    (define virtual-experimental-results [actuator_yfp riboj00_part_ribozyme ybit cysm])
    virtual-experimental-results))
```

riboj00_part _ribozyme and actuator_yfp are dependent. We use the same cluster id we sampled previously to determine mean and standard deviation for the gaussian component.

We then sample a value for riboj00_part _ribozyme from an accordingly parameterized Gaussian component.

```
(define generate-virtual-experimental-results-using-model-1
  (gen []

    (define cluster-for-actuator_yfp-and-riboj00_part_ribozyme (
        categorical [0.62 0.29 0.09]))

    (define [actuator_yfp-mean actuator_yfp-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [34278.55 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [336058.53125 432304.475202]))
    (define actuator_yfp (gaussian actuator_yfp-mean actuator_yfp-std))

    (define [riboj00_part_ribozyme-mean riboj00_part_ribozyme-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [83284.60 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [0.0 0.01]))
    (define riboj00_part_ribozyme
        (gaussian riboj00_part_ribozyme-mean riboj00_part_ribozyme-std))

    (define cluster-for-ybit-and-cysm (categorical [0.59 0.24 0.09 0.05 0.02 0.01]))

    (define [ybit-mean ybit-std]  (cond
      (= cluster-for-ybit-and-cysm 0) [149.01 46.92]
      (= cluster-for-ybit-and-cysm 1) [72.21 15.56]
      (= cluster-for-ybit-and-cysm 2) [185.27 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [762.128173828 0.01]
      (= cluster-for-ybit-and-cysm 5) [0.0 0.01]))
    (define ybit (gaussian ybit-mean ybit-std))

    (define [cysm-mean cysm-std] (cond
      (= cluster-for-ybit-and-cysm 0) [150.76 46.92]
      (= cluster-for-ybit-and-cysm 1) [43.56 15.56]
      (= cluster-for-ybit-and-cysm 2) [641.10 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 5) [813.07 420.35]))
    (define cysm (gaussian cysm-mean cysm-std))

    (define virtual-experimental-results [actuator_yfp riboj00_part_ribozyme ybit cysm])
    virtual-experimental-results))
```

The learned model indicates that ybit and cysm are dependent variables; but independent from actuator_yfp and riboj00_part_ribozyme.

We sample a new, different cluster id for ybit and cysm.

```
(define generate-virtual-experimental-results-using-model-1
  (gen []

    (define cluster-for-actuator_yfp-and-riboj00_part_ribozyme (
        categorical [0.62 0.29 0.09]))

    (define [actuator_yfp-mean actuator_yfp-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [34278.55 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [336058.53125 432304.475202]))
    (define actuator_yfp (gaussian actuator_yfp-mean actuator_yfp-std))

    (define [riboj00_part_ribozyme-mean riboj00_part_ribozyme-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [83284.60 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [0.0 0.01]))
    (define riboj00_part_ribozyme
        (gaussian riboj00_part_ribozyme-mean riboj00_part_ribozyme-std))

    (define cluster-for-ybit-and-cysm (categorical [0.59 0.24 0.09 0.05 0.02 0.01]))

    (define [ybit-mean ybit-std]  (cond
      (= cluster-for-ybit-and-cysm 0) [149.01 46.92]
      (= cluster-for-ybit-and-cysm 1) [72.21 15.56]
      (= cluster-for-ybit-and-cysm 2) [185.27 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [762.128173828 0.01]
      (= cluster-for-ybit-and-cysm 5) [0.0 0.01]))
    (define ybit (gaussian ybit-mean ybit-std))

    (define [cysm-mean cysm-std] (cond
      (= cluster-for-ybit-and-cysm 0) [150.76 46.92]
      (= cluster-for-ybit-and-cysm 1) [43.56 15.56]
      (= cluster-for-ybit-and-cysm 2) [641.10 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 5) [813.07 420.35]))
    (define cysm (gaussian cysm-mean cysm-std))

    (define virtual-experimental-results [actuator_yfp riboj00_part_ribozyme ybit cysm])
    virtual-experimental-results))
```

Repeat the process from above and draw values from one Gaussian for ybit and from another for cysm with parameters that depend on the value of cluster-for-ybit-and-cysm.

```
(define generate-virtual-experimental-results-using-model-1
  (gen []

    (define cluster-for-actuator_yfp-and-riboj00_part_ribozyme (
        categorical [0.62 0.29 0.09]))

    (define [actuator_yfp-mean actuator_yfp-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [34278.55 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [336058.53125 432304.475202]))
    (define actuator_yfp (gaussian actuator_yfp-mean actuator_yfp-std))

    (define [riboj00_part_ribozyme-mean riboj00_part_ribozyme-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [83284.60 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [0.0 0.01]))
    (define riboj00_part_ribozyme
        (gaussian riboj00_part_ribozyme-mean riboj00_part_ribozyme-std))

    (define cluster-for-ybit-and-cysm (categorical [0.59 0.24 0.09 0.05 0.02 0.01]))

    (define [ybit-mean ybit-std]  (cond
      (= cluster-for-ybit-and-cysm 0) [149.01 46.92]
      (= cluster-for-ybit-and-cysm 1) [72.21 15.56]
      (= cluster-for-ybit-and-cysm 2) [185.27 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [762.128173828 0.01]
      (= cluster-for-ybit-and-cysm 5) [0.0 0.01]))
    (define ybit (gaussian ybit-mean ybit-std))

    (define [cysm-mean cysm-std] (cond
      (= cluster-for-ybit-and-cysm 0) [150.76 46.92]
      (= cluster-for-ybit-and-cysm 1) [43.56 15.56]
      (= cluster-for-ybit-and-cysm 2) [641.10 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 5) [813.07 420.35]))
    (define cysm (gaussian cysm-mean cysm-std))

    (define virtual-experimental-results [actuator_yfp riboj00_part_ribozyme ybit cysm])
    virtual-experimental-results))
```

```
(define generate-virtual-experimental-results-using-model-1
  (gen []

    (define cluster-for-actuator_yfp-and-riboj00_part_ribozyme (
        categorical [0.62 0.29 0.09]))

    (define [actuator_yfp-mean actuator_yfp-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [34278.55 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [336058.53125 432304.475202]))
    (define actuator_yfp (gaussian actuator_yfp-mean actuator_yfp-std))

    (define [riboj00_part_ribozyme-mean riboj00_part_ribozyme-std] (cond
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 0) [83284.60 63904.74]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 1) [0.0 0.01]
      (= cluster-for-actuator_yfp-and-riboj00_part_ribozyme 2) [0.0 0.01]))
    (define riboj00_part_ribozyme
        (gaussian riboj00_part_ribozyme-mean riboj00_part_ribozyme-std))

    (define cluster-for-ybit-and-cysm (categorical [0.59 0.24 0.09 0.05 0.02 0.01]))

    (define [ybit-mean ybit-std]  (cond
      (= cluster-for-ybit-and-cysm 0) [149.01 46.92]
      (= cluster-for-ybit-and-cysm 1) [72.21 15.56]
      (= cluster-for-ybit-and-cysm 2) [185.27 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [762.128173828 0.01]
      (= cluster-for-ybit-and-cysm 5) [0.0 0.01]))
    (define ybit (gaussian ybit-mean ybit-std))

    (define [cysm-mean cysm-std] (cond
      (= cluster-for-ybit-and-cysm 0) [150.76 46.92]
      (= cluster-for-ybit-and-cysm 1) [43.56 15.56]
      (= cluster-for-ybit-and-cysm 2) [641.10 302.73]
      (= cluster-for-ybit-and-cysm 3) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 4) [0.0 0.01]
      (= cluster-for-ybit-and-cysm 5) [813.07 420.35]))
    (define cysm (gaussian cysm-mean cysm-std))

    (define virtual-experimental-results [actuator_yfp riboj00_part_ribozyme ybit cysm])
    virtual-experimental-results))
```

We return the virtual experiment results, i.e. the sampled values for:

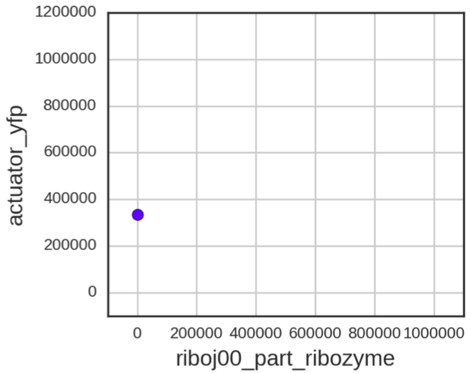
- actuator_yfp
- riboj00_part_ribozyme
- ybit
- cysm

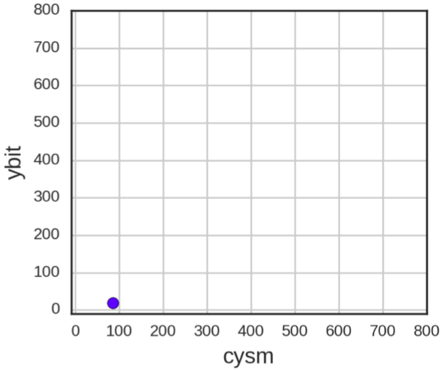
Execution of this program generates virtual data

```
(generate-virtual-experimental-results-using-model-1)
```

```
[335730.39654 0.006075 84.843161 18.668906]
```

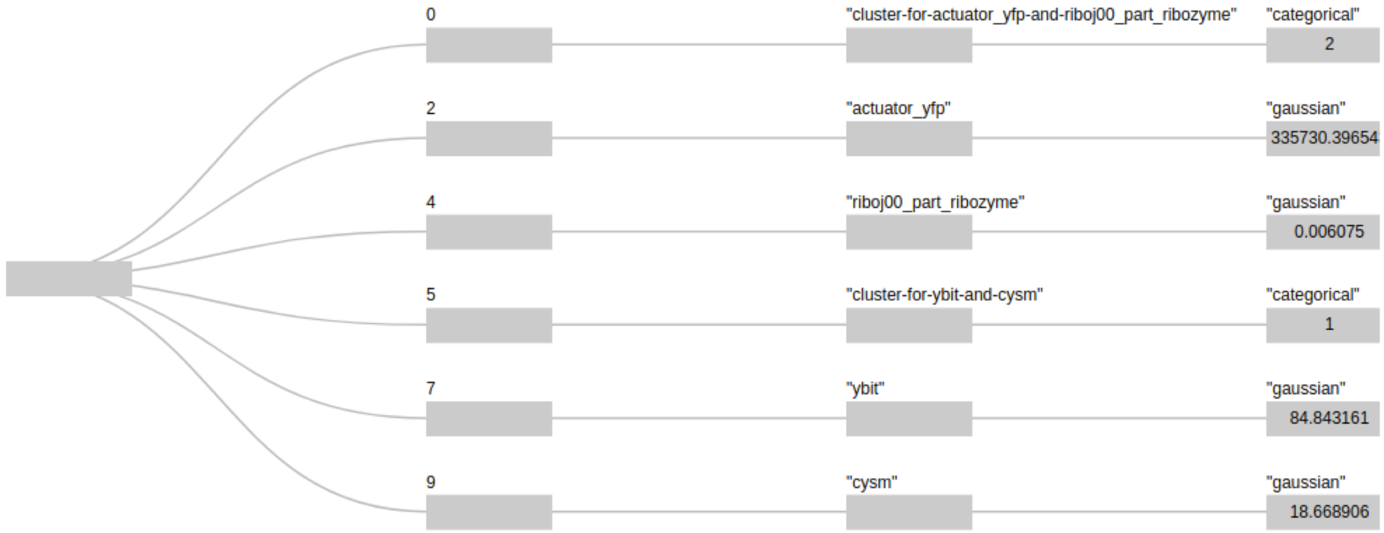
Virtual data





Execution trace of virtual experiment

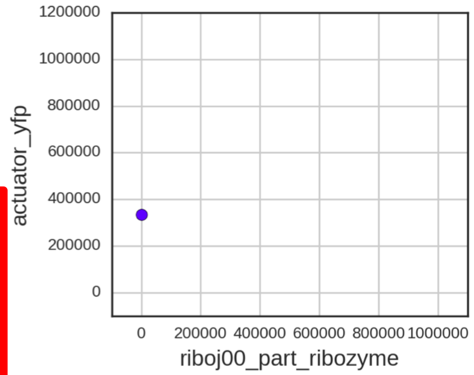
```
(define [_ execution-trace _] (infer :procedure generate-virtual-experimental-results-using-model-1 ))
(plot-trace execution-trace)
```

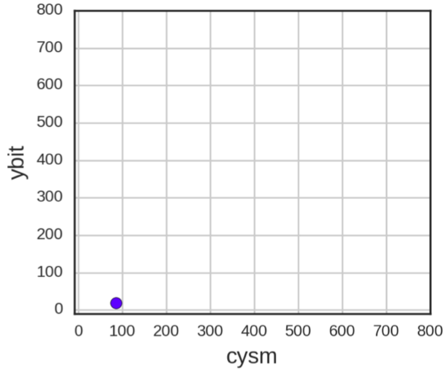


Execution of this program generates virtual data

```
(generate-virtual-experimental-results-using-model-1)

[335730.39654 0.006075 84.843161 18.668906]
```

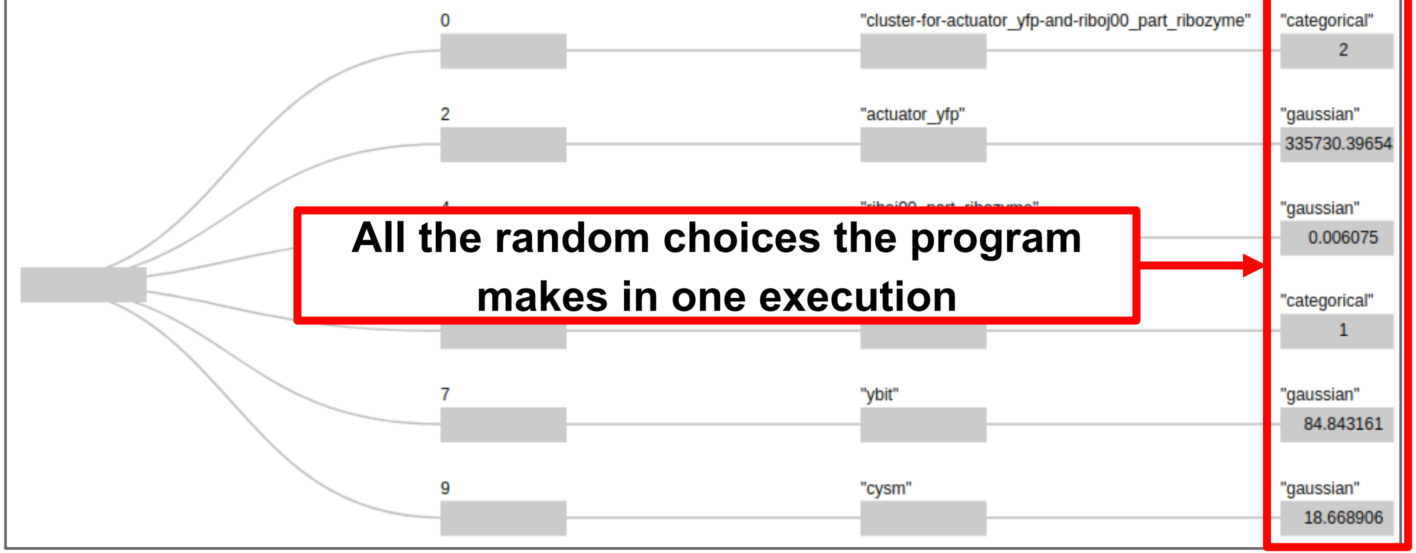
Virtual data





Execution trace of virtual experiment

```
(define [_ execution-trace _] (infer :procedure generate-virtual-experimental-results-using-model-1 ))
(plot-trace execution-trace)
```

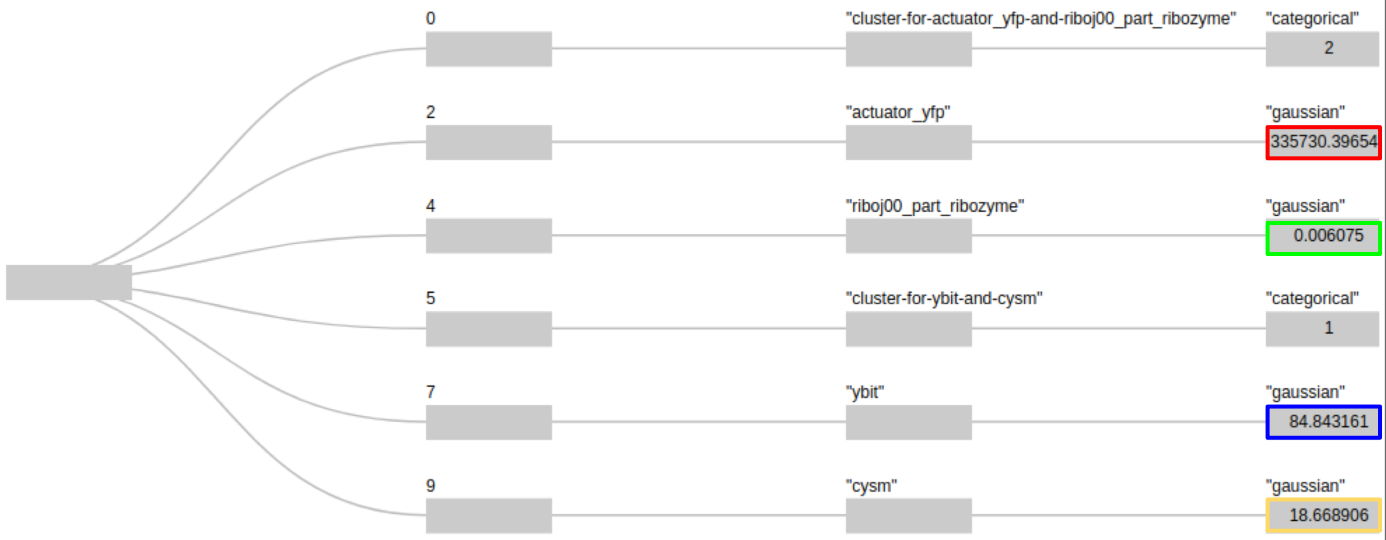


Execution of this program generates virtual data

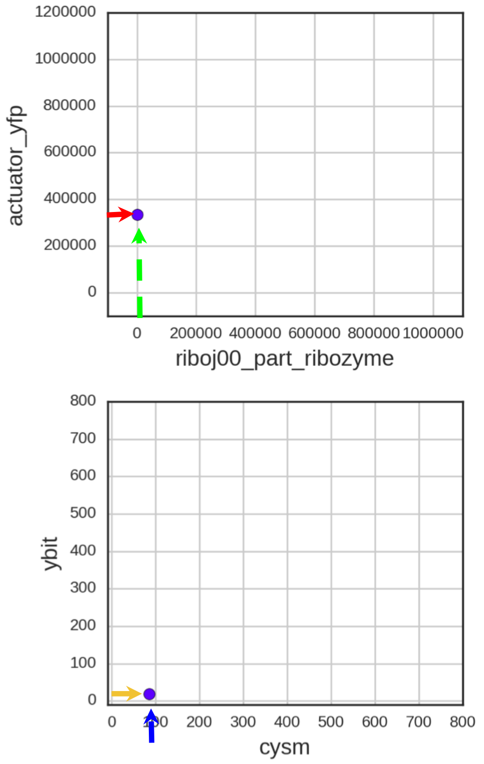
```
(generate-virtual-experimental-results-using-model-1)

[335730.39654 0.006075 84.843161 18.668906]
```

Execution trace of virtual experiment

```
(define [_ execution-trace _] (infer :procedure generate-virtual-experimental-results-using-model-1 ))
(plot-trace execution-trace)
```



Virtual data

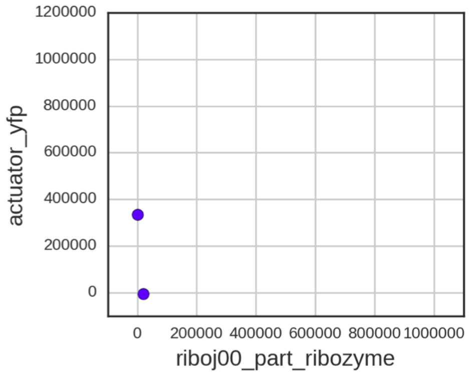


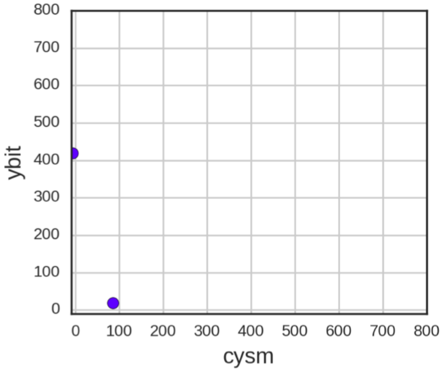
Execution of this program generates virtual data

```
(generate-virtual-experimental-results-using-model-1)

     [335730.39654 0.006075 84.843161 18.668906]
```

```
(generate-virtual-experimental-results-using-model-1)

     [-5311.874034 20137.425728 418.947872 -6.874273]
```

Virtual data





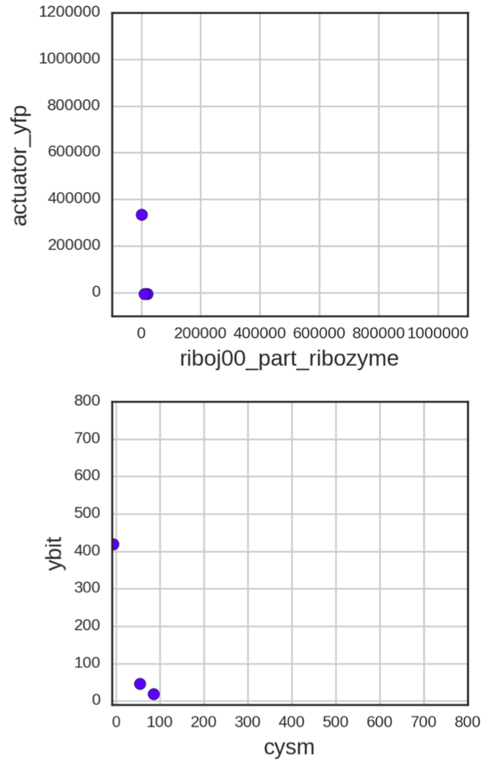
Execution of this program generates virtual data

```
(generate-virtual-experimental-results-using-model-1)

      [335730.39654 0.006075 84.843161 18.668906]
```

```
(generate-virtual-experimental-results-using-model-1)

      [-5311.874034 20137.425728 418.947872 -6.874273]
```

```
(generate-virtual-experimental-results-using-model-1)

      [-3967.569575 10886.226517 54.636702 46.125753]
```

Virtual data



Execution of this program generates virtual data

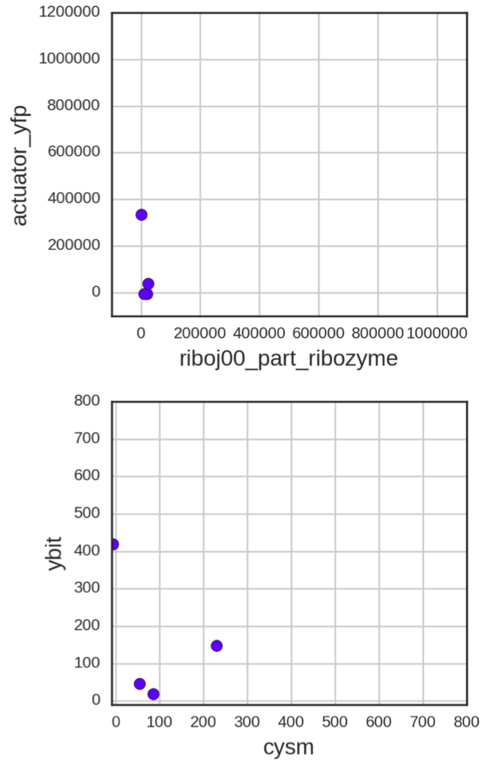
```
(generate-virtual-experimental-results-using-model-1)

      [335730.39654 0.006075 84.843161 18.668906]
```

```
(generate-virtual-experimental-results-using-model-1)

      [-5311.874034 20137.425728 418.947872 -6.874273]
```

```
(generate-virtual-experimental-results-using-model-1)

      [-3967.569575 10886.226517 54.636702 46.125753]
```

```
(generate-virtual-experimental-results-using-model-1)

      [38040.924380 23131.858116 230.307509 147.73948]
```

Virtual data



Execution of this program generates virtual data

```
(generate-virtual-experimental-results-using-model-1)
[335730.39654 0.006075 84.843161 18.668906]
```

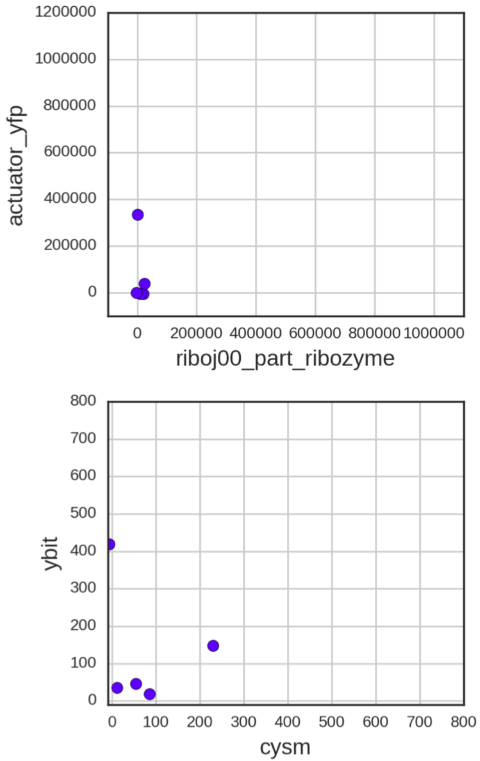
```
(generate-virtual-experimental-results-using-model-1)
[-5311.874034 20137.425728 418.947872 -6.874273]
```

```
(generate-virtual-experimental-results-using-model-1)
[-3967.569575 10886.226517 54.636702 46.125753]
```

```
(generate-virtual-experimental-results-using-model-1)
[38040.924380 23131.858116 230.307509 147.73948]
```

```
(generate-virtual-experimental-results-using-model-1)
[909.331470 -2293.825225 10.919185 36.128689]
```

Virtual data



Execution of this program generates virtual data

```
(generate-virtual-experimental-results-using-model-1)
```

[335730.39654 0.006075 84.843161 18.668906]

```
(generate-virtual-experimental-results-using-model-1)
```

[-5311.874034 20137.425728 418.947872 -6.874273]

```
(generate-virtual-experimental-results-using-model-1)
```

[-3967.569575 10886.226517 54.636702 46.125753]

```
(generate-virtual-experimental-results-using-model-1)
```

[38040.924380 23131.858116 230.307509 147.73948]

```
(generate-virtual-experimental-results-using-model-1)
```

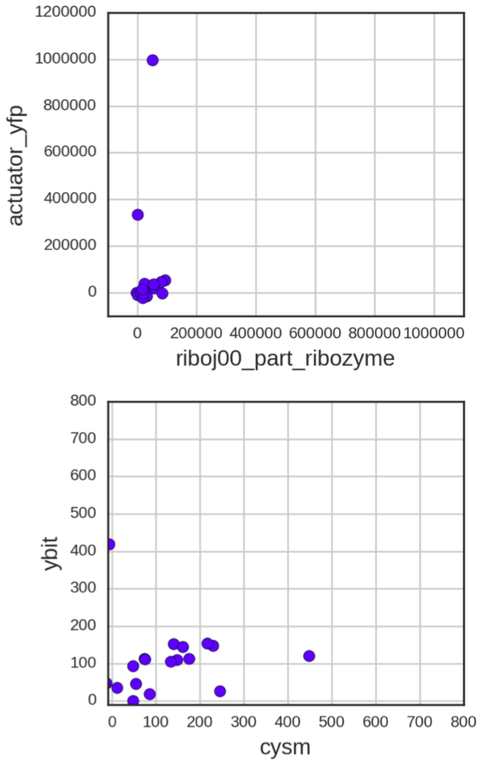
[909.331470 -2293.825225 10.919185 36.128689]

```
(generate-virtual-experimental-results-using-model-1)
```

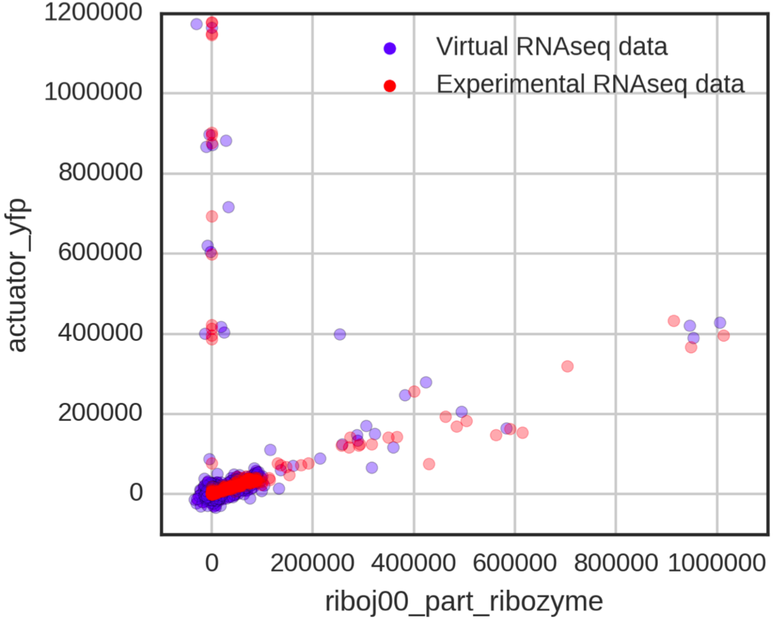
[53525.121077 93310.599669 47.987178 1.289953]

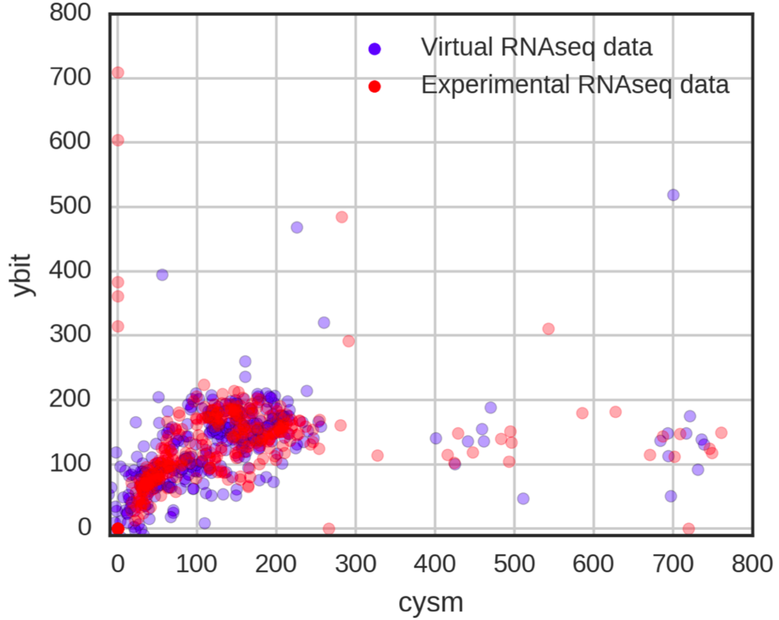
■ ■ ■

Virtual data



Compare virtual and experimental RNAseq data






```
%%bql
SIMULATE riboj00_part_ribozyme, actuator_yfp
      FROM data;
```

```
%%bql
SIMULATE cysm, ybit FROM data;
```

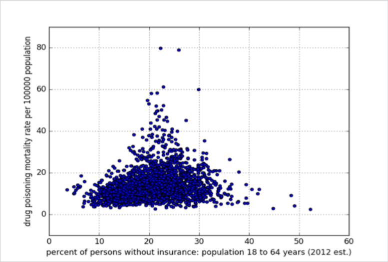
```
%%bql
SELECT riboj00_part_ribozyme, actuator_yfp
      FROM data;
```

```
%%bql
SELECT cysm, ybit FROM data;
```

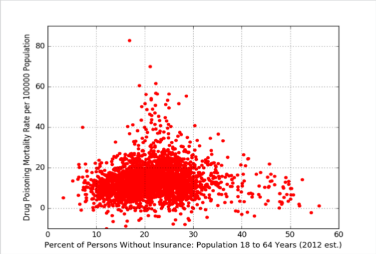
What is BayesDB?

BayesDB: An open-source probabilistic programming platform with built-in automatic model discovery.

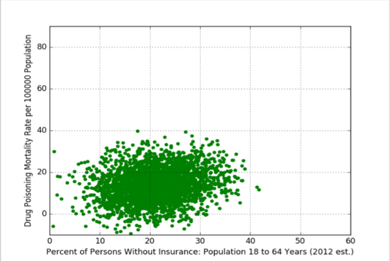
BayesDB users can use an SQL-like language to solve data analysis and statistical inference problems in seconds/minutes that otherwise take hours/days for someone with PhD-level expertise.



Real data

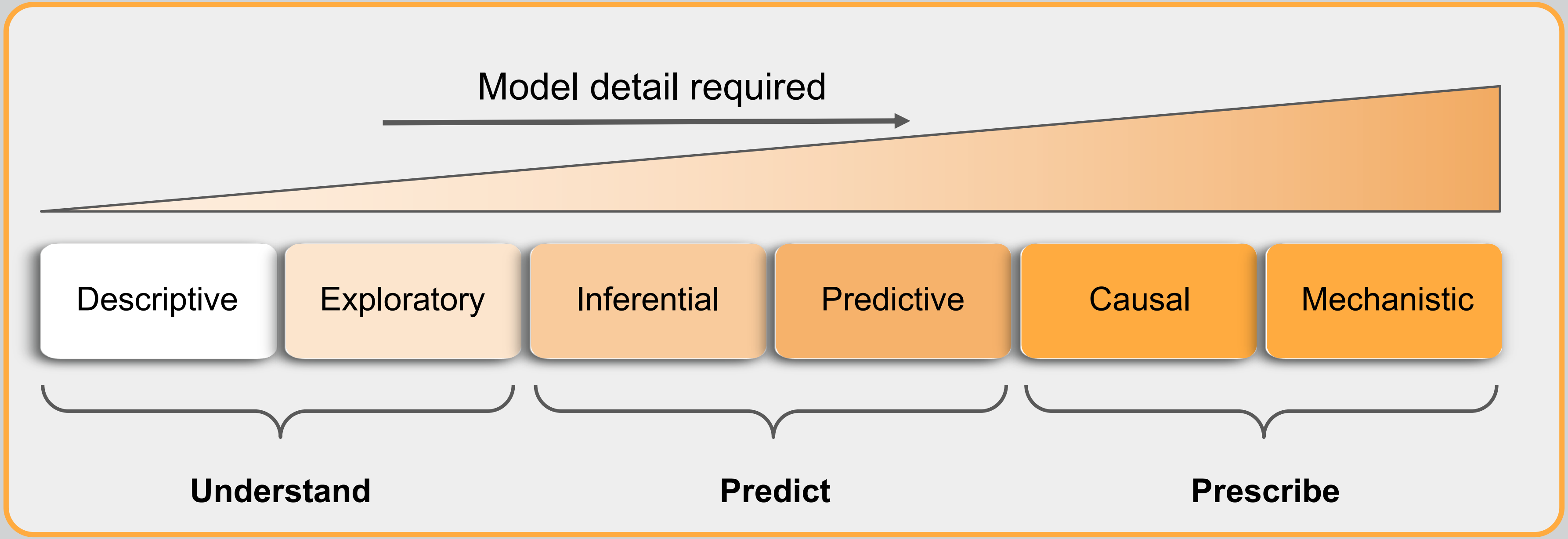


BayesDB simulations



Linear statistical model

What kinds of analysis can BayesDB perform?



NOTE: BayesDB is a research prototype, but we are currently selecting industry and government partners for a new open-source version under development

America's socio-political reality

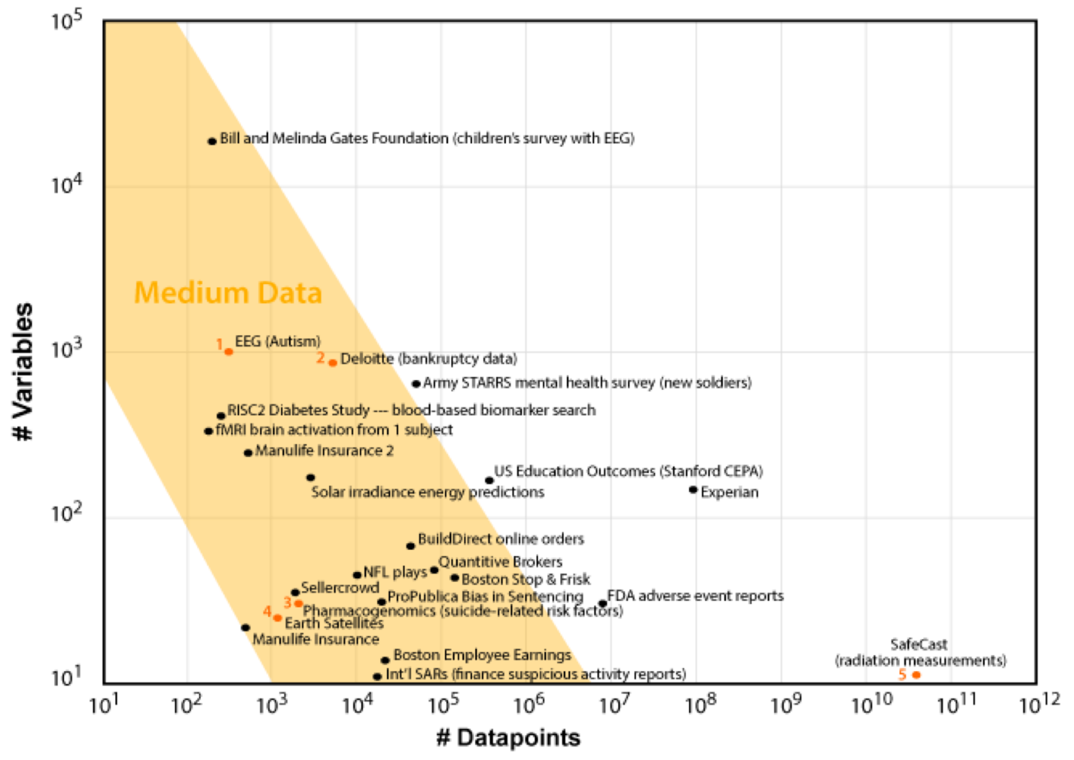
Data: empirical state of the nation

id	geo_fips	state	NAME	state_cd_slug	updated	nyt_rating	character	alex	alex_type
4	0101	al	Congressional District 1 (115th Congress), Alabama	al-01	8/6/18 10:38	1) Solid R	Rural/Small To	NULL	NULL
6	0102	al	Congressional District 2 (115th Congress), Alabama	al-02	8/6/18 10:38	1) Solid R	Rural/Small To	NULL	NULL
8	0103	al	Congressional District 3 (115th Congress), Alabama	al-03	8/6/18 10:38	1) Solid R	Rural/Small To	NULL	NULL
10	0104	al	Congressional District 4 (115th Congress), Alabama	al-04	8/6/18 10:38	1) Solid R	Rural/Small To	NULL	NULL
12	0105	al	Congressional District 5 (115th Congress), Alabama	al-05	8/6/18 10:38	1) Solid R	Rural/Small To	NULL	NULL
14	0106	al	Congressional District 6 (115th Congress), Alabama	al-06	8/6/18 10:38	1) Solid R	Mature suburt	NULL	NULL
16	0107	al	Congressional District 7 (115th Congress), Alabama	al-07	8/6/18 10:38	7) Solid D	Rural/Small To	NULL	NULL
20	0200	ak	Congressional District (at Large) (115th Congress), Ala	ak-00	8/6/18 10:38	1) Solid R	Rural/Small To	NULL	NULL
24	0401	az	Congressional District 1 (115th Congress), Arizona	az-01	8/6/18 10:38	6) Likely D	Rural/Small To	x	western_ag
26	0402	az	Congressional District 2 (115th Congress), Arizona	az-02	8/6/18 10:38	5) Lean D	Emerging subu	x	diverse
28	0403	az	Congressional District 3 (115th Congress), Arizona	az-03	8/6/18 10:38	7) Solid D	Emerging subu	NULL	NULL
30	0404	az	Congressional District 4 (115th Congress), Arizona	az-04	8/6/18 10:38	1) Solid R	Rural/Small To	NULL	NULL
32	0405	az	Congressional District 5 (115th Congress), Arizona	az-05	8/6/18 10:38	1) Solid R	Mature suburt	NULL	NULL
34	0406	az	Congressional District 6 (115th Congress), Arizona	az-06	8/6/18 10:38	2) Likely R	Mature suburt	NULL	NULL
36	0407	az	Congressional District 7 (115th Congress), Arizona	az-07	8/6/18 10:38	7) Solid D	Mature suburt	NULL	NULL

Virtual simulator of the socio-political landscape, as probabilistic program

```
(define data-generating-process-model-0
  (gen []
    (define cluster-id-for-percent_hispanic-percent_asian (categorical [0.27 0.15 0.12 0.11 0.10 0.030.03 0.15]))

    (define [percent_hispanic-mean percent_hispanic-std] (cond
      (= cluster-id-for-percent_hispanic-percent_asian 0) [0.153391 0.077936]
      (= cluster-id-for-percent_hispanic-percent_asian 1) [0.030235 0.011154]
      (= cluster-id-for-percent_hispanic-percent_asian 2) [0.130182 0.054384]
      (= cluster-id-for-percent_hispanic-percent_asian 3) [0.067915 0.020329]
      (= cluster-id-for-percent_hispanic-percent_asian 4) [0.437521 0.152850]
      (= cluster-id-for-percent_hispanic-percent_asian 5) [0.719667 0.082805]
      (= cluster-id-for-percent_hispanic-percent_asian 6) [0.237667 0.085933]
      (= cluster-id-for-percent_hispanic-percent_asian 7) [0.175795 0.180930]))
    (define percent_hispanic (gaussian percent_hispanic-mean percent_hispanic-std))

    (define [percent_asian-mean percent_asian-std] (cond
      (= cluster-id-for-percent_hispanic-percent_asian 0) [0.035929 0.011853]
      (= cluster-id-for-percent_hispanic-percent_asian 1) [0.011578 0.004625]
```

Example applications of BayesDB



Focus on “medium data”:

- 100 - 1M records
- 10 - 1000 fields

Sources of “medium data”:

- People
- Experiments
- New business processes
- “Big data” reduced down to just what's relevant

Outline

1. **Motivation**
2. **What is probabilistic programming?**

    Pedagogical example: simple (or not-so-simple) curve fitting

3. **Programmable inference, not just black-box**

    Application: machine perception via inverse graphics

4. **Learning the structure and parameters of probabilistic programs**

    Application: automatic data modeling for scientific data analysis

5. **The MIT Modeling and Inference Stack**

The MIT Modeling and Inference Stack

Gen: combining generative models, neural nets, optimization, and Monte Carlo

Perception for robotics
Analyzing scientific images
Research on common-sense AI

BayesDB: SQL-like queries and automatic data modeling

Screening databases for errors and potential anomalies
Searching databases interactively
Detecting predictive relationships from sparse data

Metaprob: lightweight, embedded probabilistic programming in Clojure

Cloudless: containerized deployment and distributed inference

The MIT Modeling and Inference Stack

Gen : combining generative models, neural nets, optimization, and Monte Carlo

Perception for robotics
Analyzing scientific images
Research on common-sense AI

Email vkm@mit.edu for info on field testing in 2019

BayesDB: SQL-like queries and automatic data modeling

Screening databases for errors and potential anomalies
Searching databases interactively
Detecting predictive relationships from sparse data

Metaprob: lightweight, embedded probabilistic programming in Clojure

Cloudless: containerized deployment and distributed inference